Sorgente : http://holsi.hasanaj.com/ Corso di Ingegneria Informatica Roma Tre
Autore : Holsi Hasanaj Corso: Analisi e Progettazione del Software
Professore: Luca Cabibbo Anno di produzione: 2012 - 2013

Analisi e Progettazione del Software

Appunti-Riassunti

Analisi e progettazione software 04/03/12

Lezione I

ricevimento MARTEDÌ 14:00 - 15:30

APS : user
UML : password

registrarsi al corso per superare l'esame con le prove in itinere

PATTERN: non bisogna sempre vedere sempre le stesse cose, un pattern è una coppia problema soluzione, rende efficace la comunicazione.

Analisi → capire quale è il problema → ragiona ad un livello di progettazione concettuale → non è INTERESSATA ALLE SOLUZIONI
"CHE COSA"

Progettazione → è anche scienza → enfatizza la soluzione
"COME"

modello → è una semplificazione della realtà che descrive completamente un sistema da un particolare punto di vista
- l'importanza dei modelli sono le IDEE e come vengono comunicate queste idee

UML : è una notazione standard per descrivere software OO
↳ visuale per scrivere (disegnare) modelli
↳ le valore primario della modellazione è nella discussione durante la modellazione

ANALISI DELLE
RESPONSABILITÀ : obbligo dato ad una attività, prima si capisce le responsabilità quali sono e poi assegnarle agli oggetti
- impegno a eseguire un compito o di conoscere un'informazione

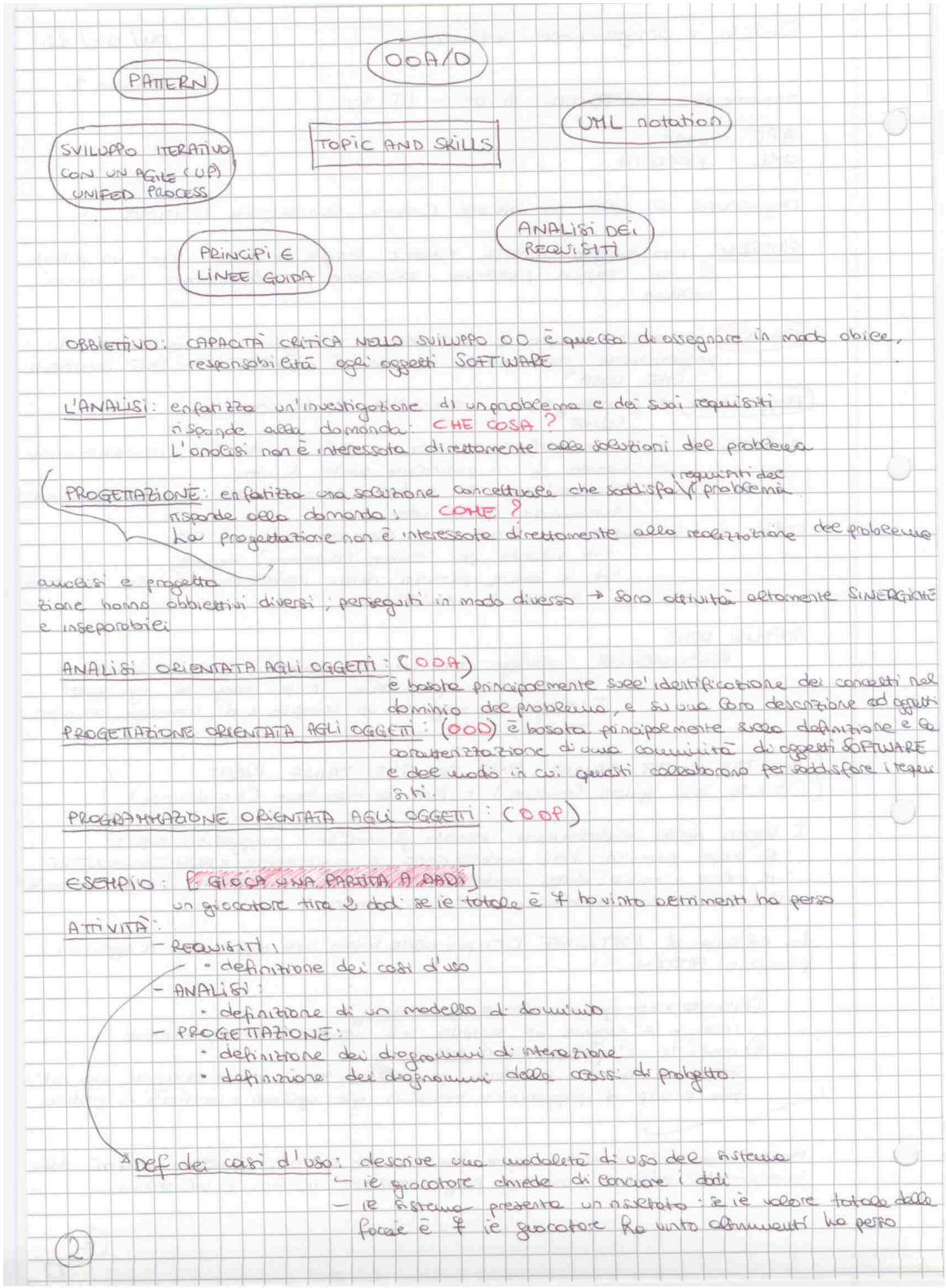
{ DO THE RIGHT THING, AND DO THE THING RIGHT
(fare la cosa giusta (analisi), e fare la cosa bene (progettazione) }

I segreti della modellazione secondo gli analisti e progettisti esperti
- lo scopo primario della modellazione è comprendere e NON DOCUMENTARE
- il valore primario della modellazione è nella discussione durante la modellazione → noi modelliamo per poter conversare

Lo sviluppo di SOFTWARE OO può essere basato sulle applicazione di opportuni principi e PATTERN:

1) progettazione guidata alle responsabilità
impegno a eseguire un compito o di conoscere un'informazione
2) mediante l'applicazione di pattern,
soluzione a un problema progettuale semplice ed eleganti, che codifica principi di progettazione esemplari, già applicati e verificati in pratica

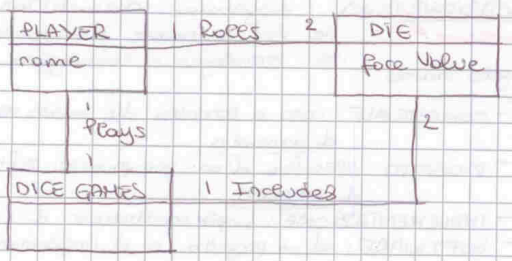
molto importante è anche capire che relazione c'è con le altre attività dello sviluppo del software: requisiti programmazione verifica e validazione


OOA/D
PATTERN
UML notation
SVILUPPO ITERATIVO CON UN AGILE (UP) UNIFED PROCESS
TOPIC AND SKILLS
ANALISI DEI REQUISITI
PRINCIPI E LINEE GUIDA
OBBIETTIVO: CAPACITÀ CRITICA NELLO SVILUPPO OO È QUELLA DI ASSEGNARE IN MODO OBIEE, RESPONSABILITÀ AGLI OGGETTI SOFTWARE
L'ANALISI: enfatizza un'investigazione di un problema e dei suoi requisiti
risponde alla domanda: CHE COSA ?
L'analisi non è interessata direttamente alle soluzioni del problema
PROGETTAZIONE: enfatizza una soluzione concettuale che soddisfa i requisiti del problema
risponde alla domanda: COME ?
la progettazione non è interessata direttamente alla realizzazione del problema
analisi e progetta
zione hanno obbiettivi diversi; perseguiti in modo diverso → sono attività altamente SINERGICHE e inseparabili
ANALISI ORIENTATA AGLI OGGETTI: (OOA)
è basata principalmente sull'identificazione dei concetti nel dominio del problema, e su una loro descrizione ad oggetti
PROGETTAZIONE ORIENTATA AGLI OGGETTI: (OOD) è basata principalmente sulla definizione e la caratterizzazione di una collettività di oggetti SOFTWARE e dei modi in cui questi collaborano per soddisfare i requisiti.
PROGRAMMAZIONE ORIENTATA AGLI OGGETTI: (OOP)
ESEMPIO: [GIOCA UNA PARTITA A DADI]
un giocatore tira 2 dadi se ie totale è 7 ho vinto altrimenti ha perso
ATTIVITÀ:
- REQUISITI:
  - definizione dei casi d'uso
- ANALISI:
  - definizione di un modello di dominio
- PROGETTAZIONE:
  - definizione dei diagrammi di interazione
  - definizione dei diagrammi delle classi di progetto
Def dei casi d'uso: descrive una modalità di uso del sistema
- ie giocatore chiede di lanciare i dadi
- ie sistema presenta un risultato: se ie valore totale della facce è 7 ie giocatore ha vinto altrimenti ha perso
2

Analisi e progettazione software 04/03/12

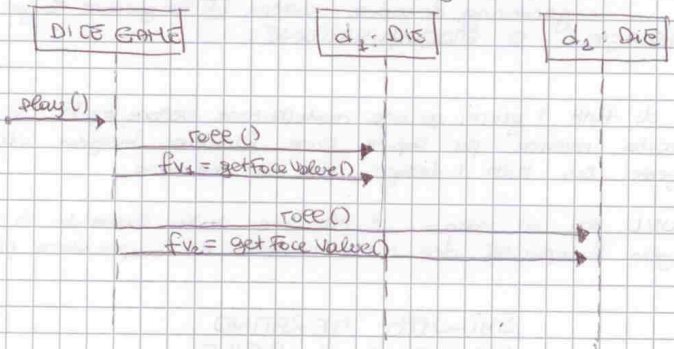
Lezione 3

- DEFINIZIONE DI UN MODELLO DI DOMINIO:
rappresenta graficamente i concetti, le associazioni e gli attributi significativi del dominio di interesse

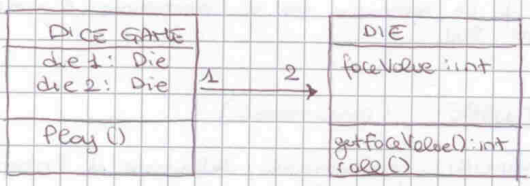


! qui si parla di oggetti del mondo reale

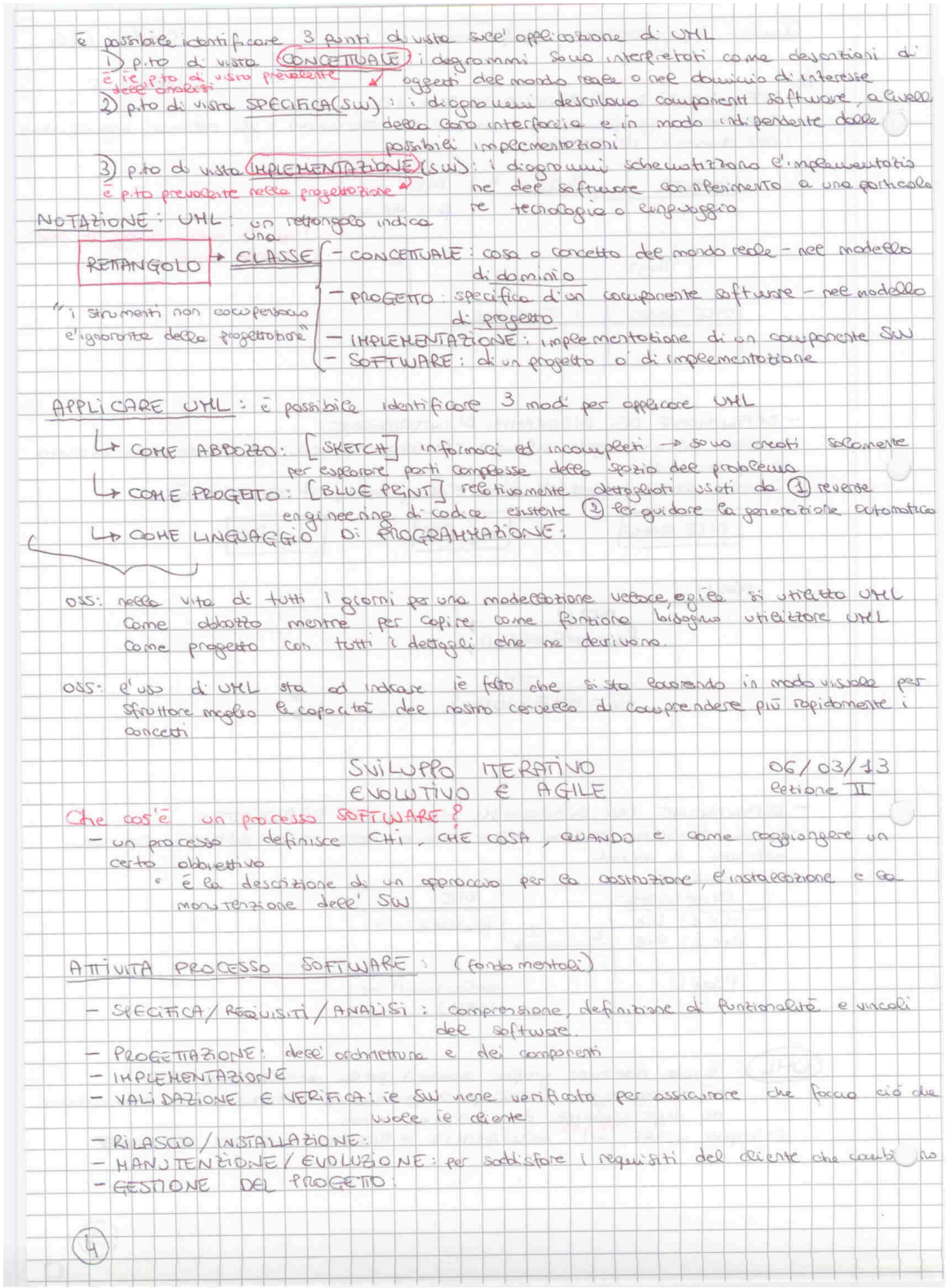
- DEFINIZIONE DEI DIAGRAMMI DI INTERAZIONE:
mostra alcuni oggetti software e le loro collaborazioni, per ottenere un certo comportamento
Descrive scelte progettuali circa l'assegnazione di responsabilità



- DEFINIZIONE DEI DIAGRAMMI DELLE CLASSI DI PROGETTO:
è una descrizione statica della struttura delle classi software con i loro attributi e metodi



UML: è una notazione grafica standard per la modellazione OO
→ è un linguaggio visuale per la specifica, la costruzione e la documentazione dei degli elaborati di un sistema e non
BISOGNA USARE (APPLICARE) UML NELL'ANALISI E PROGETTAZIONE DEL SOFTWARE



È possibile identificare 3 punti di vista sull'applicazione di UML

1) p.to di vista CONCETTUALE: i diagrammi sono interpretati come descrizioni di oggetti del mondo reale o nel dominio di interesse
è il p.to di vista prevalente nelle analisi

2) p.to di vista SPECIFICA (SW): i diagrammi descrivono componenti software, a livello della loro interfaccia e in modo indipendente dalle possibili implementazioni

3) p.to di vista IMPLEMENTAZIONE (SW): i diagrammi schematizzano l'implementazione del software con riferimento a una particolare tecnologia o linguaggio
è p.to prevalente nella progettazione

NOTAZIONE: UML un rettangolo indica una

RETTANGOLO → CLASSE
- CONCETTUALE: cosa o concetto del mondo reale - nel modello di dominio
- PROGETTO: specifica di un componente software - nel modello di progetto
- IMPLEMENTAZIONE: implementazione di un componente SW
- SOFTWARE: di un progetto o di implementazione

"i strumenti non compensano l'ignoranza della progettazione"

APPLICARE UML: è possibile identificare 3 modi per applicare UML

↳ COME ABBOZZO: [SKETCH] informali ed incompleti → sono creati selezionate per esplorare parti complesse della spazio del problema
↳ COME PROGETTO: [BLUE PRINT] relativamente dettagliati usati da (1) reverse engineering di codice esistente (2) per guidare la generazione automatica
↳ COME LINGUAGGIO DI PROGRAMMAZIONE:

OSS: nella vita di tutti i giorni per una modellazione veloce, agile si utilizza UML come abbozzo mentre per capire come funziona bisogna utilizzare UML come progetto con tutti i dettagli che ne derivano.

OSS: l'uso di UML sta ad indicare il fatto che si sta ragionando in modo visuale per sfruttare meglio la capacità del nostro cervello di comprendere più rapidamente i concetti

SVILUPPO ITERATIVO EVOLUTIVO E AGILE

06/03/13
Lezione II

Che cos'è un processo SOFTWARE?
- un processo definisce CHI, CHE COSA, QUANDO e come raggiungere un certo obiettivo
  - è la descrizione di un approccio per la costruzione, l'installazione e la manutenzione del SW

ATTIVITÀ PROCESSO SOFTWARE: (fondamentali)
- SPECIFICA/REQUISITI/ANALISI: comprensione, definizione di funzionalità e vincoli del software.
- PROGETTAZIONE: dell'architettura e dei componenti
- IMPLEMENTAZIONE
- VALIDAZIONE E VERIFICA: il SW viene verificato per assicurare che faccia ciò che vuole il cliente
- RILASCIO/INSTALLAZIONE:
- MANUTENZIONE/EVOLUZIONE: per soddisfare i requisiti del cliente che cambiano
- GESTIONE DEL PROGETTO:

Analisi e Progettazione Software

06/03/13
lezione II

Perché esistono diversi processi software?
- i processi software si differenziano soprattutto nel QUANDO, ovvero nel modo di determinare l'ordine delle attività

UNIFIED PROCESS: è uno specifico processo per lo sviluppo di software OO (un framework di processo, ben documentato)
↳ viene utilizzato perché il contesto in cui si applica al meglio è un processo iterativo applicato in modo agile
- inoltre è aperto e flessibile ed incoraggia l'uso di pratiche
  - come EXTREME PROGRAMMING (XP), SCRUM

↳ i processi, tecnologie sono importanti, ma sono le persone che li mettono in pratica → LE PERSONE SONO MOLTO PIÙ IMPORTANTI

PROCESSO A CASCATA: un progetto software classico è basato su un ciclo di vita SEQUENZIALE LINEARE e prevede
- ciascuna attività produce documenti dettagliati e lavora producendo documenti a quella successiva solo con l'approvazione dei documenti
  - PIANIFICAZIONE: con stime dettagliate per tutte le attività
  - ANALISI DEI REQUISITI SOFTWARE
  - PROGETTAZIONE
  - GENERAZIONE DEL CODICE
  - COLLAUDO

Cosa significa sequenziale?
- faccio prima tutta l'analisi dei requisiti
- poi faccio tutta l'analisi
- poi faccio tutta la progettazione
- poi scrivo il codice
- poi inizio a fare il collaudo
- solo a quest. p.to ho qualcosa di fun

DIFETTI: è spesso poco efficace nello sviluppo del software — perché sono coinvolte le persone
Il processo a cascata non consente l'introduzione di buone idee intraprese, in quanto per come è fatto il processo bisognerebbe ripartire da capo

↳ soggetto a fallimenti perché non consente una gestione efficace dei rischi inoltre è molto costoso gestire errori introdotti nelle prime fasi

HP PROCESSO A CASCATA: deve essere possibile definire correttamente e "completamente" i requisiti prima di procedere con le fasi successive

OSS: il vero problema quando si deve sviluppare un progetto è che il committente chiede al progettista A anche se non sa che vorrebbe B, il progettista invece pensa di aver capito e sviluppa C
Ecco perché nei progetti bisogna essere molto flessibili

SVILUPPO ITERATIVO: lo sviluppo del SOFTWARE è organizzato in una serie di mini-progetti brevi di lunghezza fissa chiamate ITERAZIONI dove
- il risultato di ciascuna iterazione è un sistema software eseguibile, testato ma PARZIALE
- ciascuna iterazione comprende le proprie attività di analisi dei requisiti etc
- il sistema cresce in modo incrementale da un'iterazione alla successiva adattandosi ai requisiti in modo evolutivo sulla base del FEEDBACK delle iterazioni precedenti.

(!) nello sviluppo ITERATIVO ciascuna iterazione opera su un piccolo sottoinsieme di requisiti : analisi, progettazione, implementazione e verifica
→ fornisce un ritorno (FEED BACK) rapido : utenti, sviluppatori, TEST
questo ritorno è un'opportunità per far crescere la misura di ridurre i rischi
in particolare produrre codice ed esporlo ai clienti

IDEA DI FONDO : anticipare nel tempo parte di alcune attività e RIMANDARE nel tempo parte di altre attività, cercando di massimizzare i benefici e minimizzare i rischi
ad esempio la comprensione di requisiti meno importanti — non si può procedere a caso la pianificazione è importante

VANTAGGI : - minor probabilità di fallimento del progetto, miglior produttività, percentuali più basse di difetti
- riduzione precoce anziché tardiva dei rischi maggiori
- visibilità del progresso
- feedback precoce, impegno degli utenti, adattamento
- gestione della complessità

SVANTAGGI - dire di adottare un processo iterativo ma pensare di in cascata

TIMEBOXING : ciascuna iterazione ha una durata prefissata (1-6 settimane)
la durata di un'iterazione, una volta fissata non può cambiare
(ecco perché Time boxed)
→ è meglio ridurre i requisiti di un'iterazione piuttosto che non rispettarne la scadenza

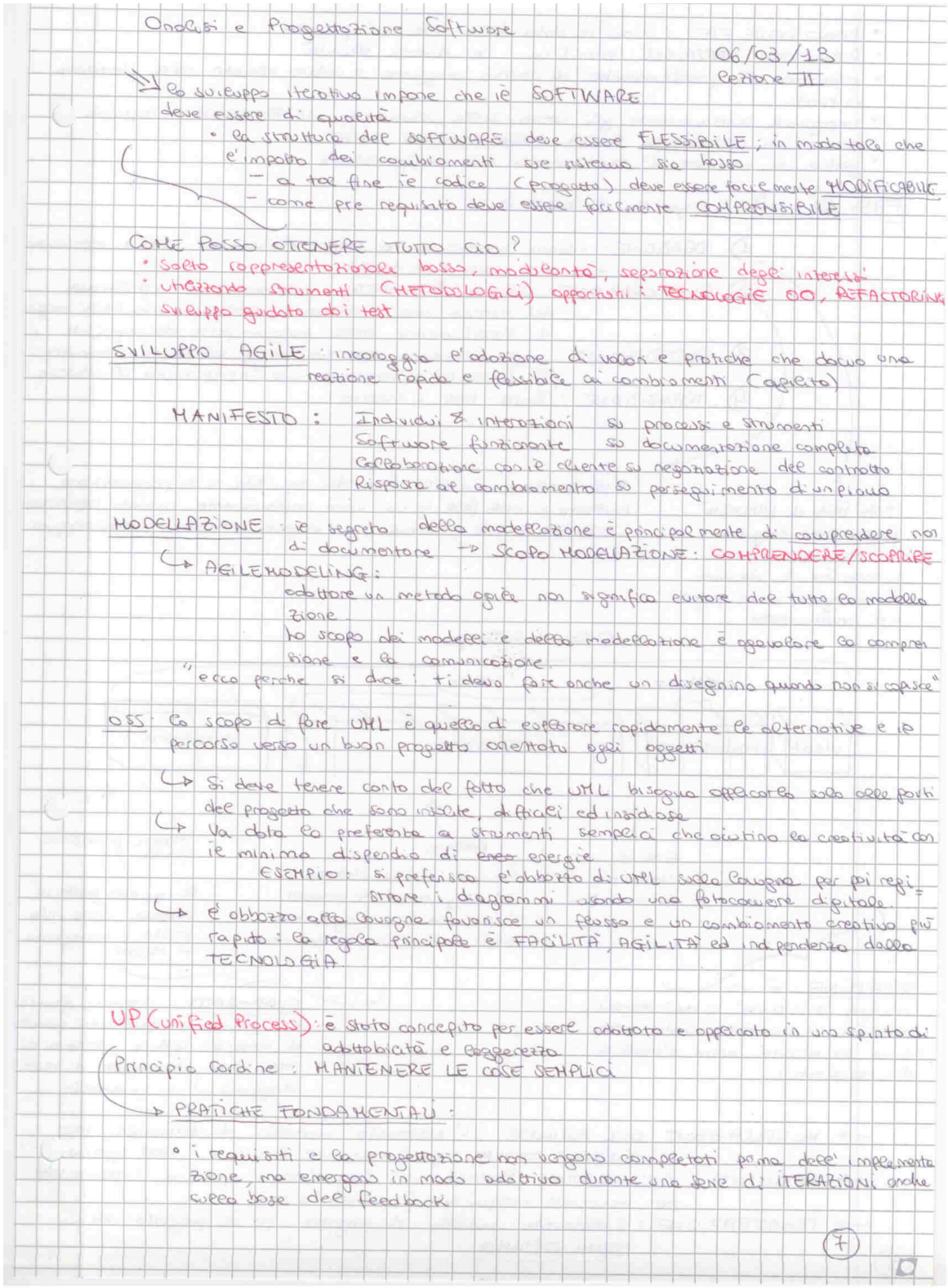
OSS : durante ciascuna iterazione, i requisiti su cui operare vengono prefissati e BLOCCATI così che al termine di ciascuna iterazione i committenti possano vedere il progresso effettivo dello sviluppo e fornire un feedback per guidare lo sviluppo successivo

BACKLOG : elenco arretrato è uno strumento per l'organizzazione del lavoro che deve essere ancora svolto nello sviluppo iterativo
→ è un elenco dei requisiti che devono essere ancora implementati o problemi ancora da risolvere
→ anche la gestione del BACKLOG è iterativo → le voci del backlog vengono aggiornate su anche sulla base del valore di business e dei rischi associati a quella voce

PRINCIPIO DI PARETO : la maggior parte degli effetti è dovuta a un numero ristretto di cause
- LEGGE 80/20 -
la maggior parte degli effetti (80%) è dovuta a un numero ristretto di cause
le prime iterazioni si occupano del 20% dei requisiti o del sistema
(!) ma non di un 20% a caso → del 20% dei requisiti che contribuiscono all'80% del valore complessivo

PIANIFICAZIONE ITERATIVA : pratica fondamentale dello sviluppo iterativo, ovvero risponde alla domanda : CHE COSA FARE NELLA PROSSIMA ITERAZIONE ?

- UP incoraggia pianificazione iterazione guidata dal RISCHIO e guidata dal CLIENTE
→ infatti l'obiettivo delle prime iterazioni è : identificare e affrontare i rischi principali
- SVILUPPO CENTRATO SULL'ARCHITETTURA : le prime iterazioni si concentrano sulla costruzione di TEST e la stabilizzazione del NUCLEO dell'architettura.



Analisi e Progettazione Software

06/03/13
Lezione II

Lo sviluppo iterativo impone che il SOFTWARE deve essere di qualità
- la struttura del SOFTWARE deve essere FLESSIBILE; in modo tale che l'impatto dei cambiamenti sul sistema sia basso
  - a tal fine il codice (progetto) deve essere facilmente MODIFICABILE
  - come pre requisito deve essere facilmente COMPRENSIBILE

COME POSSO OTTENERE TUTTO CIO?
- Scelta rappresentazionale basso, modularità, separazione degli interessi
- Utilizzando strumenti (METODOLOGICI) opportuni: TECNOLOGIE OO, REFACTORING, sviluppo guidato dai test

SVILUPPO AGILE: incoraggia l'adozione di valori e pratiche che danno una reazione rapida e flessibile ai cambiamenti (agile)

MANIFESTO:
Individui & interazioni su processi e strumenti
Software funzionante su documentazione completa
Collaborazione con il cliente su negoziazione del contratto
Risposta al cambiamento su perseguimento di un piano

MODELLAZIONE: il segreto della modellazione è principalmente di comprendere non di documentare → SCOPO MODELLAZIONE: COMPRENDERE/SCOPRIRE
↳ AGILE MODELING:
adottare un metodo agile non significa evitare del tutto la modellazione
lo scopo dei modelli e della modellazione è agevolare la comprensione e la comunicazione
"ecco perche si dice: ti devo fare anche un disegnino quando non si capisce"

OSS: Lo scopo di fare UML è quello di esplorare rapidamente le alternative e il percorso verso un buon progetto orientato agli oggetti
↳ Si deve tenere conto del fatto che UML bisogna applicarlo solo alle parti del progetto che sono inutili, difficili ed instabili
↳ Va data la preferenza a strumenti semplici che aiutino la creatività con il minimo dispendio di energie
ESEMPIO: si preferisce l'abbozzo di UML sulla lavagna per più persone disegnare i diagrammi usando una fotocamera digitale
↳ L'abbozzo alla lavagna favorisce un feedback e un cambiamento creativo più rapido; la regola principale è FACILITÀ, AGILITÀ ed indipendenza dalla TECNOLOGIA

UP (Unified Process): è stato concepito per essere adottato e applicato in uno spirito di adattabilità e leggerezza
Principio cardine: MANTENERE LE COSE SEMPLICI
↳ PRATICHE FONDAMENTALI:
- i requisiti e la progettazione non vengono completati prima dell'implementazione, ma emergono in modo adattivo durante una serie di ITERAZIONI anche sulla base dei feedback

7

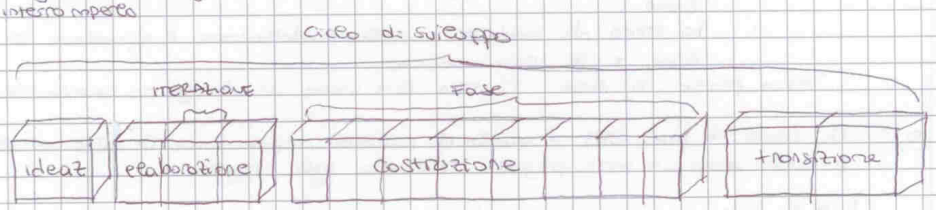
- PIANO DELLE FASI : stima la data della fine del progetto
- PIANO DELL'ITERAZIONE: pianifica le diverse attività
- affrontare le problematiche di rischio maggiore e valore eseguito nelle iterazioni iniziali
- impegnare continuamente gli utenti sulla valutazione, le FEEDBACK e i requisiti
- creare un architettura SOLIDA nelle iterazioni iniziali
- gestire le richieste di cambiamento e le configurazioni

FASI DI UP: organizza il lavoro in:

1) IDEAZIONE: visione approssimativa, studio economico, stime approssimative dei costi e dei tempi
2) ELABORAZIONE: visione RAFFINATA, implementazione ITERATIVA dell'nucleo dell'architettura, risoluzione dei rischi maggiori, identificazione della maggior parte dei requisiti
3) COSTRUZIONE: implementazione iterativa degli elementi rimanenti più facili e a rischio minore, preparazione al rilascio
4) TRANSIZIONE: Beta-test e rilascio

è la fase di fattibilità in cui viene eseguita un'indagine sufficiente a sostenere la decisione di proseguire con il progetto o di interromperlo

l'elaborazione non è la fase dei requisiti o di progettazione, piuttosto è una fase in cui viene implementata in modo ITERATIVO l'architettura del sistema e vengono mitigati i rischi maggiori



fine di un'iterazione in cui si verifica

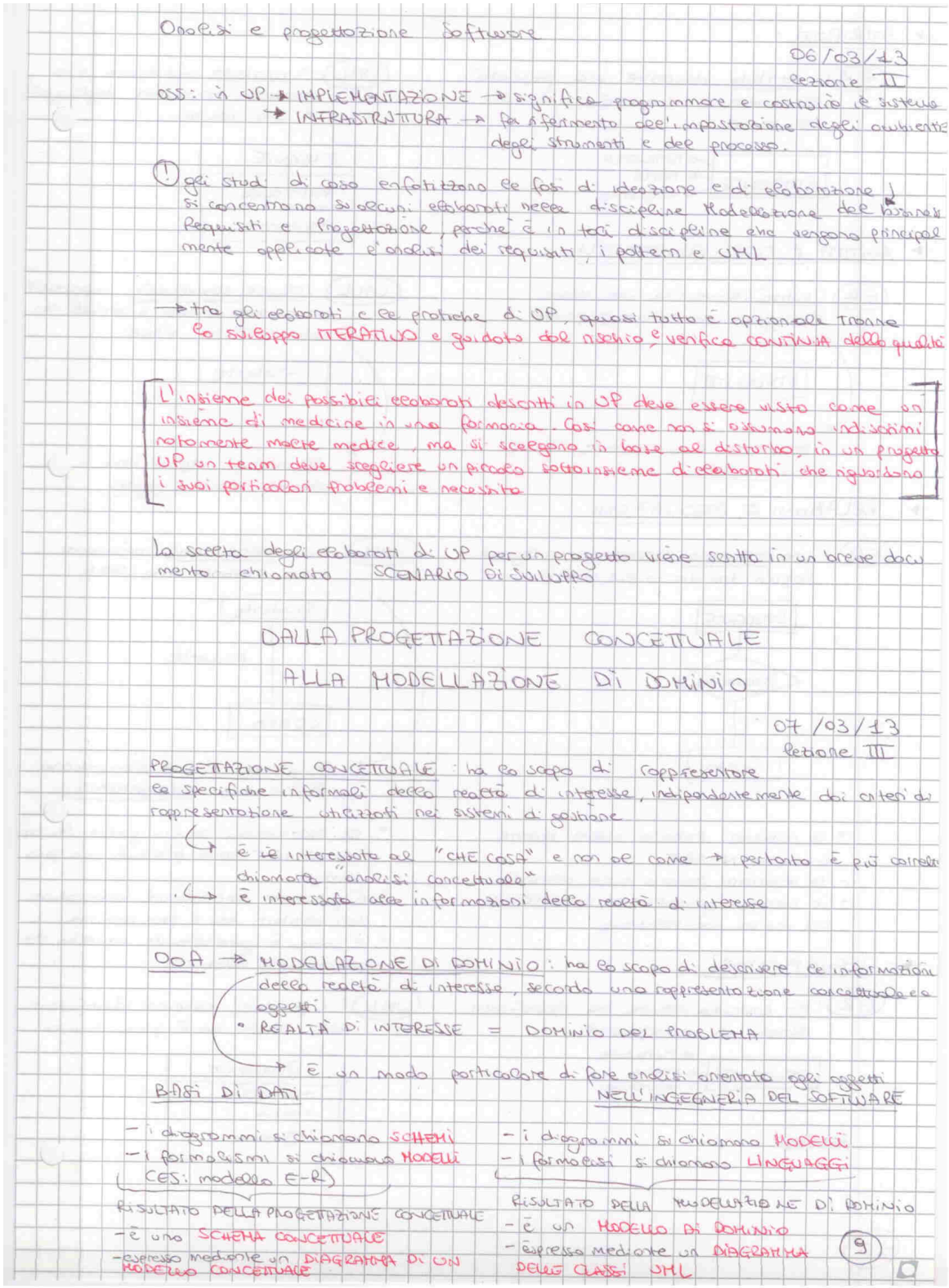
Release
Un sottoinsieme stabile ed eseguibile del prodotto finale. La fine di ogni iterazione produce una release minore

release produzione finale
a questo punto il sistema viene rilasciato e consegnato ai clienti

DISCIPLINE: è un insieme di attività e dei relativi elaborati in una determinata area (come le attività entro l'analisi dei requisiti)

ELABORATO: termine generico che indica un qualsiasi prodotto di lavoro: codice, schemi di basi dati, documenti di testo, diagrammi e modelli

→ MODELLAZIONE DEL BUSINESS: l'elaborato modello di dominio, per visualizzare i concetti significativi nel dominio dell'applicazione
→ REQUISITI: gli elaborati Modello dei Casi d'uso e Specifica Supplementare per descrivere i requisiti funzionali e non
→ PROGETTAZIONE: l'elaborato modello di progetto, per il progetto degli oggetti software



Analisi e progettazione Software

06/03/13
lezione II

oss: in UP → IMPLEMENTAZIONE → significa programmare e costruire il sistema
→ INFRASTRUTTURA → fa riferimento all'impostazione degli ambienti degli strumenti e del processo.

① gli studi di caso enfatizzano le fasi di ideazione e di elaborazione → si concentrano su alcuni elaborati nelle discipline Rodelizzazione dei Requisiti e Progettazione, perché è in tali discipline che vengono principalmente applicate l'analisi dei requisiti, i pattern e UML

→ tra gli elaborati e le pratiche di UP, quasi tutto è opzionale TRANNE lo sviluppo ITERATIVO e guidato dal rischio e verifica CONTINUA della qualità

L'insieme dei possibili elaborati descritti in UP deve essere visto come un insieme di medicine in una farmacia. Così come non si assumono indistintamente tutte medicine, ma si scelgono in base al disturbo, in un progetto UP un team deve scegliere un piccolo sottoinsieme di elaborati che riguardano i suoi particolari problemi e necessità

La scelta degli elaborati di UP per un progetto viene scritta in un breve documento chiamato SCENARIO di sviluppo

DALLA PROGETTAZIONE CONCETTUALE ALLA MODELLAZIONE DI DOMINIO

07/03/13
lezione III

PROGETTAZIONE CONCETTUALE: ha lo scopo di rappresentare le specifiche informali della realtà di interesse, indipendentemente dai criteri di rappresentazione utilizzati nei sistemi di gestione
↳ è interessata al "CHE COSA" e non al come → pertanto è più corretto chiamarla "analisi concettuale"
↳ è interessata alle informazioni della realtà di interesse

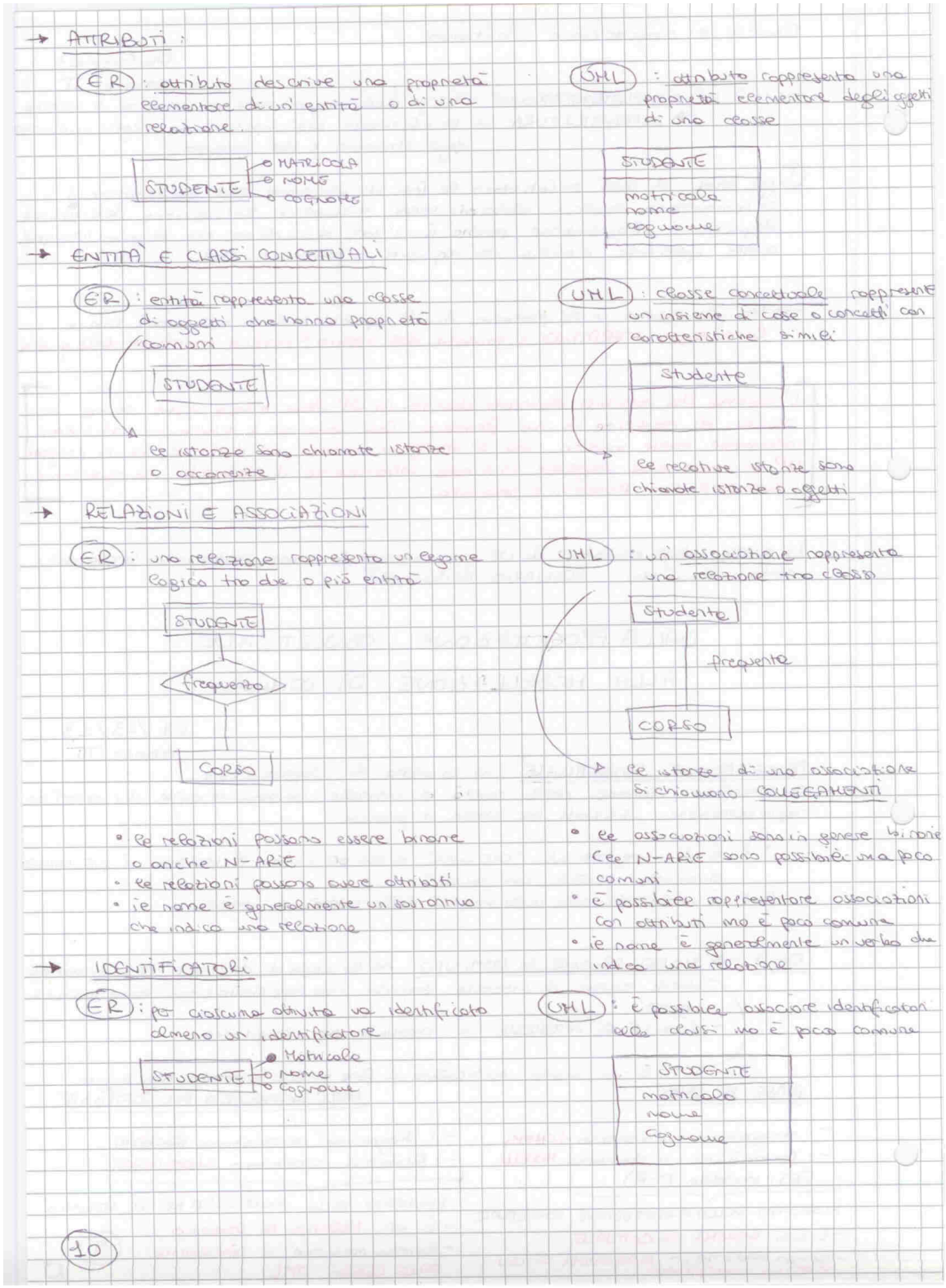
OOA → MODELLAZIONE DI DOMINIO: ha lo scopo di descrivere le informazioni della realtà di interesse, secondo una rappresentazione concettuale e a oggetti
- REALTÀ DI INTERESSE = DOMINIO DEL PROBLEMA
→ è un modo particolare di fare analisi orientata agli oggetti

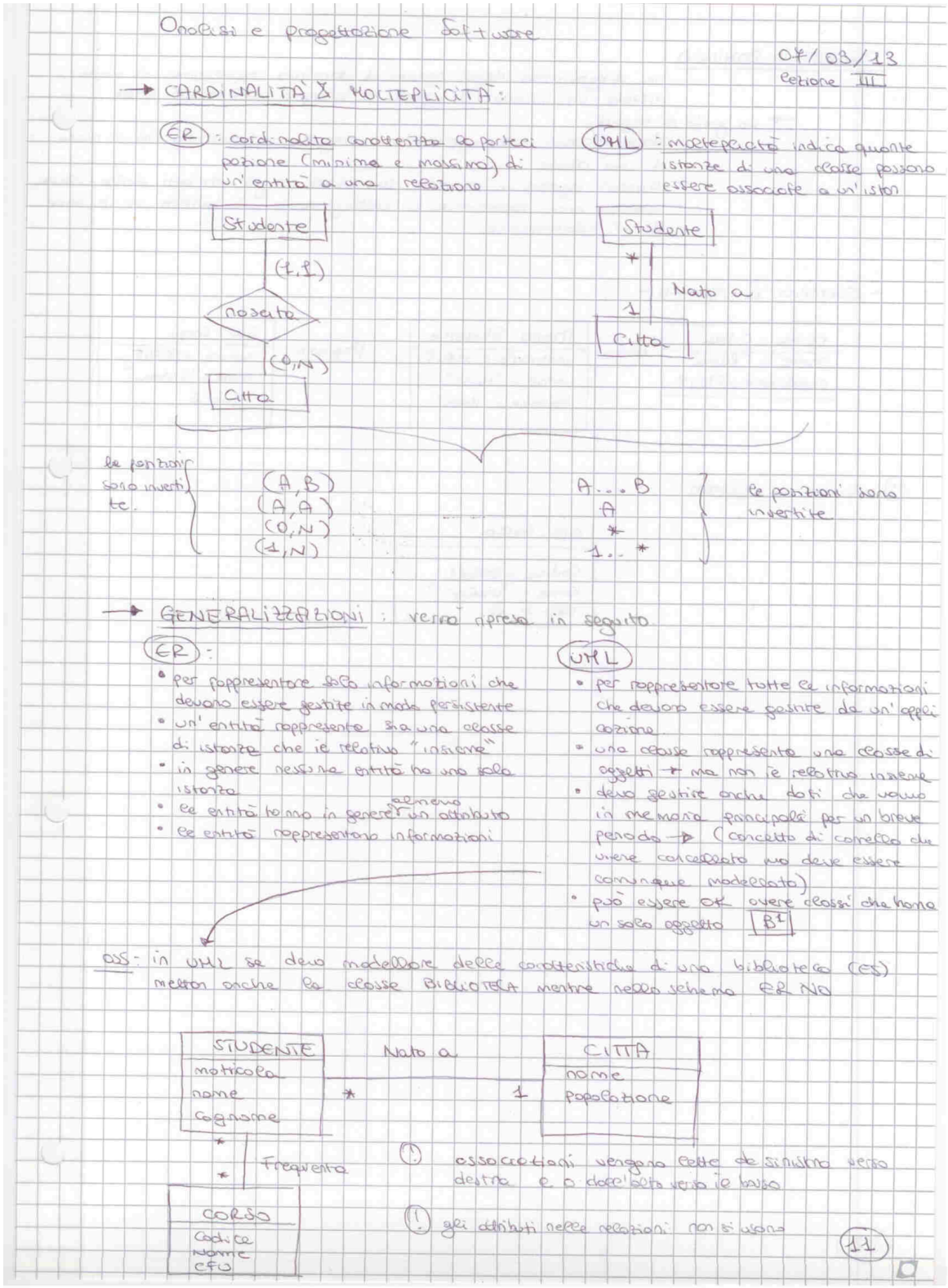
BASI DI DATI
- i diagrammi si chiamano SCHEMI
- i formalismi si chiamano MODELLI (es: modello E-R)

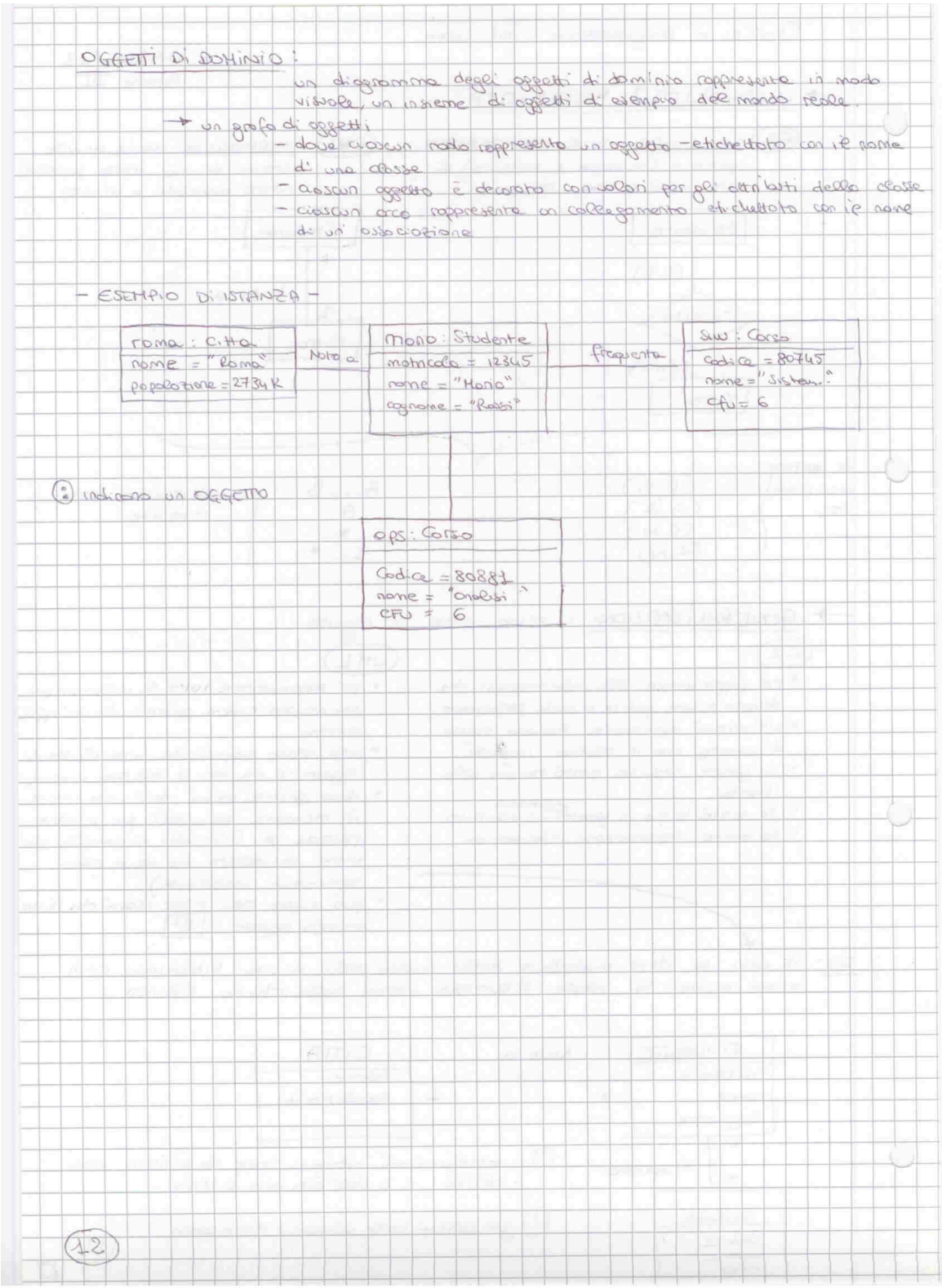
RISULTATO DELLA PROGETTAZIONE CONCETTUALE
- è uno SCHEMA CONCETTUALE
- espresso mediante un DIAGRAMMA DI UN MODELLO CONCETTUALE

NELL'INGEGNERIA DEL SOFTWARE
- i diagrammi si chiamano MODELLI
- i formalismi si chiamano LINGUAGGI

RISULTATO DELLA MODELLAZIONE DI DOMINIO
- è un MODELLO DI DOMINIO
- espresso mediante un DIAGRAMMA DELLE CLASSI UML


ATTRIBUTI
ER: attributo descrive una proprietà elementare di un'entità o di una relazione.
STUDENTE
MATRICOLA
NOME
COGNOME
UML: attributo rappresenta una proprietà elementare degli oggetti di una classe
STUDENTE
matricola
nome
cognome
ENTITÀ E CLASSI CONCETTUALI
ER: entità rappresenta una classe di oggetti che hanno proprietà comuni
STUDENTE
le istanze sono chiamate istanze o occorrenze
UML: classe concettuale rappresenta un insieme di cose o concetti con caratteristiche simili
Studente
le relative istanze sono chiamate istanze o oggetti
RELAZIONI E ASSOCIAZIONI
ER: una relazione rappresenta un legame logico tra due o più entità
STUDENTE
frequento
CORSO
UML: un associazione rappresenta una relazione tra classi
Studente
frequenta
CORSO
le istanze di una associazione si chiamano COLLEGAMENTI
le relazioni possono essere binarie o anche N-arie
le relazioni possono avere attributi
il nome è generalmente un sostantivo che indica una relazione
le associazioni sono in genere binarie (le N-arie sono possibili ma poco comuni
è possibile rappresentare associazioni con attributi ma è poco comune
il nome è generalmente un verbo che indica una relazione
IDENTIFICATORI
ER: per ciascuna entità va identificato almeno un identificatore
STUDENTE
Matricola
nome
cognome
UML: è possibile associare identificatori alla classe ma è poco comune
STUDENTE
matricola
nome
cognome
10


Analisi e progettazione software
07/03/13
Lezione III
CARDINALITÀ & MOLTEPLICITÀ:
ER: cardinalità caratterizza la partecipazione (minima e massima) di un'entità a una relazione
Studente
(1,1)
nascita
(0,N)
Città
UML: molteplicità indica quante istanze di una classe possono essere associate a un'istanza
Studente
*
Nato a
1
Città
le posizioni sono invertite
(A,B)
(A,A)
(0,N)
(1,N)
A...B
A
*
1..*
le posizioni sono invertite
GENERALIZZAZIONI: verrà ripresa in seguito
ER:
per rappresentare solo informazioni che devono essere gestite in modo persistente
un'entità rappresenta una classe di istanze che ie relativi "insieme"
in genere nessuna entità ha una sola istanza
ee entità hanno in genere almeno un attributo
ee entità rappresentano informazioni
UML
per rappresentare tutte ee informazioni che devono essere gestite da un'applicazione
una classe rappresenta una classe di oggetti ma non ie relativi insieme
deve gestire anche dati che vanno in memoria principale per un breve periodo → (concetto di corretto che viene concettato non deve essere comunque modellato)
può essere OK avere classi che hanno un solo oggetto BL
OSS: in UML se devo modellare delle caratteristiche di una biblioteca (es) metto anche la classe BIBLIOTECA mentre nello schema ER NO
STUDENTE
matricola
nome
cognome
Nato a
CITTÀ
nome
popolazione
Frequenta
CORSO
codice
nome
cfu
associazioni vengono lette da sinistra verso destra e o dall'alto verso il basso
gli attributi nelle relazioni non si usano
11


OGGETTI DI DOMINIO:
un diagramma degli oggetti di dominio rappresenta in modo visuale un insieme di oggetti di esempio del mondo reale
→ un grafo di oggetti
- dove ciascun nodo rappresenta un oggetto -etichettato con il nome di una classe
- ciascun oggetto è decorato con valori per gli attributi della classe
- ciascun arco rappresenta un collegamento etichettato con il nome di un'associazione
- ESEMPIO DI ISTANZA -
roma : Città
nome = "Roma"
popolazione = 2734 K
Nato a
Mario : Studente
matricola = 12345
nome = "Mario"
cognome = "Rossi"
frequenta
SWI : Corso
codice = 80745
nome = "Sistem..."
cfu = 6
① indicano un OGGETTO
ops : Corso
codice = 80881
nome = "ops"
CFU = 6
12

Analisi e progettazione del software
VideoRental – studio di caso
23 febbraio 2011
REQUISITI

VideoRental è un sistema per la gestione del noleggio di film. Esso deve gestire tutte le informazioni relative ai film che possono essere noleggiati, ai clienti e ai noleggi.
A ciascun film sono associate informazioni come il titolo e l'anno di produzione; ad esempio, il film *Blade Runner* del 1982. A ciascun film è associato un codice univoco. I film che possono essere noleggiati sono realizzati con supporti/formati diversi; ad esempio, *DVD* e *Blu-Ray*. Per ciascun film possono esserci più copie fisiche, eventualmente in formati diversi; ad esempio, ci potrebbero essere due copie di *Blade Runner* in formato *DVD* e una in formato *Blu-Ray*. Ciascuna copia è dotata di un numero di serie univoco. Dunque, un oggetto che può essere noleggiato potrebbe essere, ad esempio, una copia di *Blade Runner* in formato *DVD*, con numero di serie 6457.
Il sistema deve conoscere, per ciascun cliente, nome, cognome e recapiti (numero di cellulare e indirizzo di posta elettronica). Ciascun cliente è dotato di una tessera, caratterizzata da un numero univoco. Al momento della consegna di una tessera, a questa viene associato un credito (ovvero, un importo monetario), che viene diminuito a fronte dei noleggi effettuati dal cliente. La tessera è dotata di una banda magnetica, sulla quale è riportato esclusivamente il numero univoco della tessera.
Il sistema deve gestire noleggi e restituzioni, sia in corso che passati, anche con informazioni sulla data e sull'orario in cui sono avvenuti.
Un cliente può effettuare un noleggio solo se è dotato di un credito non nullo. Ciascun cliente può avere in noleggio, contemporaneamente, al massimo tre copie di film. Il costo di un noleggio dipende da vari fattori, tra cui: il tipo di supporto del film, la durata effettiva del noleggio, il fatto che il film scelto sia una novità.

L'uso del sistema VideoRental è descritto principalmente dai seguenti casi d'uso (di cui è di interesse soprattutto lo scenario principale di successo):

modo di rappresentare un sistema / caso funzionale d'uso

Caso d'uso UC1: Inserimento nuovo film e copie – Attore primario: un Amministratore del Sistema.
1. L'Addetto inizia l'immissione di un nuovo film.
2. L'Addetto immette titolo, anno di produzione e codice univoco del nuovo film.
3. L'Addetto immette il codice di un tipo di supporto, per indicare l'esistenza di una nuova copia del film in quel formato. Il Sistema genera un numero di serie per quella copia e lo stampa (anche sotto forma di codice a barre) su una etichetta adesiva. L'Addetto attacca l'etichetta alla copia.

Il passo 3 viene ripetuto finché serve.
4. L'Addetto indica di aver finito. Il Sistema registra tutte le informazioni sul nuovo film.

Caso d'uso UC2: Effettua noleggio – Attore primario: un Cliente del Sistema.
1. Il Cliente si reca a uno sportello del sistema per effettuare un noleggio.
2. Il Cliente inserisce la propria tessera in un apposito lettore, e digita il proprio codice personale. Il Sistema legge il numero univoco della tessera, verifica la correttezza dei dati immessi e che il Cliente possa effettuare un noleggio.
3. Il Cliente immette il codice del tipo di supporto di cui vuole effettuare un noleggio.
4. Il Sistema mostra, per il tipo di supporto selezionato, i film che sono attualmente disponibili per il noleggio.
5. Il Cliente immette il codice del film da noleggiare e conferma la propria richiesta di noleggio. Il Sistema seleziona, tra le copie disponibili per quel film e supporto, quale sarà la copia da consegnare al cliente. Il Sistema registra le informazioni sul noleggio. Il Sistema consegna al Cliente la copia selezionata.
6. Il Cliente prende la copia, estrae la sua tessera e va via.

Analisi e progettazione del software
VideoRental – studio di caso
23 febbraio 2011
REQUISITI

Caso d'uso UC3: Effettua restituzione copia – Attore primario: un Cliente del Sistema.

1. Il Cliente si reca a uno sportello del sistema per restituire una copia di film presa in noleggio.
2. Il Cliente inserisce la propria tessera in un apposito lettore, e digita il proprio codice personale. Il Sistema legge il numero univoco della tessera, e verifica la correttezza dei dati immessi.
3. Il Cliente inserisce, in una apposita feritoia, la copia del film presa in noleggio che vuole restituire. Il Sistema legge il codice della copia restituita. Il Sistema calcola l'importo dovuto dal Cliente per il noleggio di questa copia, visualizza questo importo, lo diminuisce dal credito del Cliente, e registra l'avvenuta restituzione.
4. Il Cliente estrae la sua tessera e va via.

Regole per il calcolo del costo di un noleggio – di interesse solo per l'esercizio A5.
Il costo effettivo di un noleggio dipende da vari fattori, e precisamente:

- il tipo di supporto del film noleggiato; ad esempio, il noleggio di un DVD ha un costo giornaliero di base di 1.50 euro;
- la durata effettiva del noleggio; in generale, il costo del noleggio è dato dal costo giornaliero di base moltiplicato per la durata effettiva del noleggio (calcolata come numero di giorni arrotondato per eccesso); tuttavia, se il noleggio è durato meno di 12 ore, allora il costo del noleggio è dato dal costo giornaliero di base diviso 2;
- il fatto che il film scelto sia una novità; il noleggio di una novità ha un costo aggiuntivo di 1.00 euro (indipendentemente dalla durata del noleggio).

Analisi e progettazione del software
VideoRental – studio di caso
23 febbraio 2011
ANALISI

Esercizio A1
Fare l'analisi orientata agli oggetti per il sistema in discussione, relativamente a tutti i casi d'uso mostrati, come segue:

- Mostrare il modello di dominio.
- Mostrare un modello degli oggetti di dominio che descrive
  - le tre copie a disposizione del sistema del film *Blade Runner* (due in DVD ed una in Blu-Ray)
  - il fatto che il cliente Paolo verdi ha avuto in prestito (e poi restituito) la copia in Blu-Ray del film *Blade Runner*
  - il fatto che il cliente Mario Rossi ha attualmente in prestito la copia in Blu-Ray del film *Blade Runner*

Esercizio A2
Fare l'analisi orientata agli oggetti per il sistema in discussione, relativamente al caso d'uso UC2, come segue:

- Mostrare il diagramma di sequenza di sistema per il caso d'uso UC2.
- Mostrare il contratto di tutte le operazioni di sistema per il caso d'uso UC2.

Analisi e progettazione del software
VideoRental – studio di caso
23 febbraio 2011
PROGETTAZIONE

Ipotesi di lavoro, valide per tutti gli esercizi di progettazione.

- In tutti gli esercizi che seguono, si faccia l'ipotesi che il sistema in discussione gestisca i propri dati solo in memoria principale. Si supponga anche che durante il caso d'uso di avviamento vengano creati e caricati in memoria tutti gli oggetti le cui informazioni siano già effettivamente disponibili al momento dell'avviamento.
- Per ciascuna operazione di sistema va creato un diagramma di interazione che descrive l'interazione relativa alla trasformazione (cambiamento di stato) provocata dall'operazione di sistema. Per quanto riguarda invece le relative risposte (interrogazioni) eventualmente restituite dal sistema, se non è richiesto esplicitamente allora non bisogna mostrare nei diagrammi di interazione né il calcolo dei dati da restituire né la loro visualizzazione. Tuttavia, per le risposte del sistema, è comunque necessario verificare che i dati da restituire possano essere (facilmente) calcolati sulla base delle navigabilità disponibili tra gli oggetti (vedi anche il punto successivo).
- **Le soluzioni individuate dovranno essere compatibili (in particolare in termini di visibilità, ovvero di navigabilità delle associazioni) con le realizzazioni di tutti i casi d'uso mostrati.**
- Nei diagrammi di interazione, motivare le scelte di progetto fatte indicando i pattern GRASP e GoF applicati.
- **Nei diagrammi di interazione, mostrare in modo esplicito: tutti i messaggi scambiati tra oggetti, tutte le creazioni di oggetti e tutte le formazioni/rotture di collegamenti.**

Esercizio A3
Fare la progettazione orientata agli oggetti relativa al sistema in discussione, relativamente al caso d'uso UC2, come segue:

- mostrare i diagrammi di interazione relativi a tutte le operazioni di sistema per il caso d'uso UC2;
- mostrare il corrispondente diagramma delle classi di progetto.

Esercizio A4
Fare la progettazione orientata agli oggetti relativa al sistema in discussione, relativamente al caso d'uso UC3, come segue:

- si faccia l'ipotesi semplificativa che il costo di ciascun noleggio sia esattamente 2 euro, indipendentemente da film e tipo di supporto scelto e dalla durata effettiva del noleggio;
- mostrare i diagrammi di interazione relativi a tutte le operazioni di sistema per il caso d'uso UC3;
- mostrare il corrispondente diagramma delle classi di progetto (con un DCD separato rispetto a quello dell'esercizio A3).

Esercizio A5
Fare la progettazione orientata agli oggetti relativa al sistema in discussione, relativamente all'operazione per il calcolo dell'importo dovuto per un noleggio (parte del passo 3 del caso d'uso UC3), come segue:

- mostrare un diagramma di interazione per il calcolo dell'importo dovuto per un noleggio;
- descrivere come l'importo dovuto possa essere visualizzato sull'interfaccia grafica usando il design pattern Observer [GoF], descrivendo sia gli aspetti statici che quelli dinamici, anche mediante degli opportuni diagrammi UML.

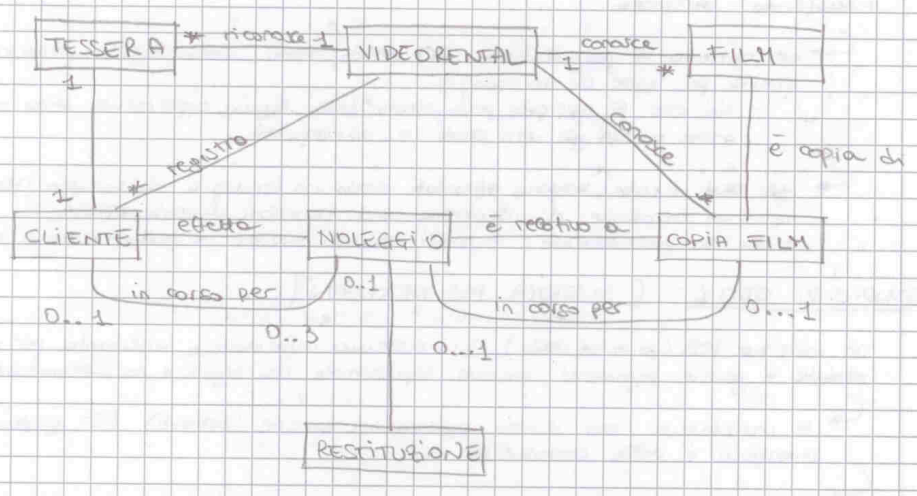
Analisi e progettazione Software

07/03/13
Lezione III

- ESERCITAZIONE VIDEO - RENTAL -

MODELLAZIONE DI DOMINIO
si parte dal concetto più importante (poiché si vuole accettare il metodo iterativo)

Nella descrizione dei requisiti di un sistema informatico le informazioni appaiono secondo casi d'uso
Una domanda che bisogna sempre farsi da progettista → (siccome è una attività di business) da dove viene il guadagno? che cos'è che mi fa guadagnare



Il noleggio è un contratto tra il cliente e il negozio (istante)
Il film è un'opera fatta da un regista attori etc è una cosa astratta

OSS: ▲ simbolo che sto ad indicare il verso di percorrenza (che dovrebbe essere da sinistra - destra ; dall'alto - basso)

OSS: una buona analisi emissi si cerca 5 risposte di perché

Come gestisco il fatto che un cliente può prendere al massimo 3 noleggi?
- posso aggiungere un attributo : in corso
- introduco un'altra associazione : in corso per

STUDI DI CASO

Lezione IV

11/03/13

Le applicazioni software sono composte da diversi elementi:

- → elementi dell'interfaccia utente
- → la logica applicativa principale
- → l'accesso alla base di dati e collaborazioni con componenti software o hardware esterni

La tecnologia OO può essere applicata a tutti i livelli ⇒ ci si focalizzerà su quello della logica applicativa

- Ci si interessa allo strato della logica APPLICATIVA nell'analisi orientata agli oggetti perché solitamente gli strati dipendono in larga misura dalla tecnologia e dalla piattaforma utilizzata

↳ al contrario la progettazione OO dello strato della logica principale è simile per tutte le tecnologie
- ciò che si apprende nello strato della logica applicativa sono applicabili anche a tutti gli altri strati o componenti

↳ gli studi di caso vengono affrontati usando una strategia di sviluppo iterativo con un'iterazione di ideazione e 3 iterazioni di elaborazione
anche la presentazione di molti concetti avviene in modo iterativo e incrementale

STUDIO DI CASO 1: [SISTEMA POS NEXTGEN]

un sistema POS (point of sale) è un sistema informatico utilizzato per registrare vendite e gestire pagamenti, presente tipicamente nei negozi e nei supermercati

↳ si interfaccia con diverse applicazioni esterne (SERVIZI) ES: gestione dell'inventario e della contabilità

↳ un sistema POS deve essere relativamente tollerante ai guasti, ciò significa che anche se i servizi remoti sono temporaneamente non disponibili: il sistema deve essere comunque in grado di gestire le vendite

↳ inoltre il POS da realizzare è un sistema commerciale, destinato a essere venduto a diversi clienti con esigenze diverse rispetto alla gestione delle regole di business → sarà necessario un meccanismo per fornire questa flessibilità e personalizzazione

ALTRI STUDI DI CASO
- VIDEO RENTAL: sistema di video noleggio
- EE edit: editor interattivo di diagrammi E-R
- Collezione di Posa
- Scale e Serpenti: versione del gioco dell'oca e del gioco del monopoly

IDEAZIONE

L'ideazione è la breve passo iniziale che permette di stabilire una visione comune e comprende: l'analisi di circa il 10% dei casi d'uso; l'analisi dei requisiti non funzionali più critici la creazione di uno studio economico e la preparazione dell'ambiente di sviluppo
domande da farsi:
- Quale è la visione e quale lo studio economico per questo progetto?
- È fattibile?
- Dovremmo procedere o fermarci?

Analisi e progettazione software

11/03/13
Lezione IV

IDEAZIONE: è l'idea è quella di effettuare un'indagine sufficiente per formarsi un'opinione razionale e giustificabile sull'obiettivo generale e sulla fattibilità del nuovo potenziale sistema

SI TRATTA DI DECIDERE SE IL PROGETTO MERITA UN'INDAGINE PIÙ SERIA

→ "Immaginare la portata del prodotto, la visione e lo studio economico"
→ ie PROBLEMA principale risolto in una frase:
"Le parti interessate hanno un'accordo di base sulla visione del progetto e vale la pena di investire di più in un'indagine seria"

La fase di ideazione che la precede è simile a uno studio di fattibilità il cui scopo è però solo di decidere se vale la pena di investire

→ dal momento in cui si tratta dell'ideazione ie contenuto dell'indagine e degli elaborati dovrebbe essere leggero
ES: ie modello dei casi d'uso può elencare i nomi della maggior parte dei casi d'uso
- E'ideazione ed può inoltre comportare un certo lavoro di programmazione volto a creare dei prototipi "PROOF OF CONCEPT"

ELABORATI PER IDEAZIONE:

VISIONE & STUDIO ECONOMICO: descrive gli obiettivi e i vincoli di alto livello, lo studio economico e fornisce un sommario del progetto

MODELLO DEI CASI D'USO: descrive i requisiti funzionali. Durante l'ideazione vengono identificati i nomi della maggior parte dei casi d'uso, e circa il 10% dei casi d'uso viene sviluppato in modo dettagliato

SPECIFICHE SUPPLEMENTARI: descrivono altri requisiti per lo più non funzionali → è utile avere un'idea dei requisiti non funzionali fondamentali che avranno un impatto significativo sull'architettura

GLOSSARIO: terminologia chiave del dominio e dizionario dei dati

! gli elaborati UP vanno considerati opzionali, bisogna scegliere di creare solo quelli che aggiungono realmente valore al progetto e scartare quelli il cui valore non è dimostrato

OSS: gli elaborati di progetti precedenti possono essere parzialmente riusati nei progetti successivi → è normale che in tutti i progetti ci siano inoltre simili gestione negli elaborati per i rischi, la gestione del progetto, ie test e l'infrastruttura

REQUISITI EVOLUTIVI:

I requisiti sono capacità e condizioni a cui ie sistema e più in generale ie progetto deve essere conforme

→ "un approccio sistematico per trovare, documentare, organizzare e tracciare i i requisiti che cambiano di un sistema

UP abbraccia il cambiamento nei requisiti come una guida fondamentale per i progetti e si inizia la programmazione di qualità e i test molto prima che la maggior parte dei requisiti siano stati utilizzati o specificati

→ bisogno tenere in mente che in media il 25% dei requisiti cambia portato questo tipo di metodo che tenti di congelare o definire completamente i requisiti fin dall'inizio è fondamentalmente difettoso

UP incoraggia un acquisizione dei requisiti agile, attraverso tecniche quali scrivere i casi d'uso con i clienti, workshop dei requisiti a cui partecipano sia sviluppatori che clienti, gruppi di lavoro con clienti delegati

MODELLO FURPS+: (Functional, usability, reliability, performance, supportability)
i requisiti in UP sono divisi in categorie:

- FUNZIONALE: caratteristiche funzionali, capacità, sicurezza
- USABILITÀ: fattori umani, help, documentazione
- AFFIDABILITÀ: frequenza di fallimento, ripristinabilità, prevedibilità
- PRESTAZIONI: tempo di risposta, throughput, precisione, disponibilità
- SOSTENIBILITÀ: adattabilità, manutenibilità, configurabilità

il segno + indica componenti complementari secondari:
→ IMPLEMENTAZIONE: limitazione sulle risorse, linguaggi e strumenti
→ INTERFACCIA: vincoli imposti nella necessità di interfacciarsi con sistemi esterni
→ LEGALI: licenze e così via.

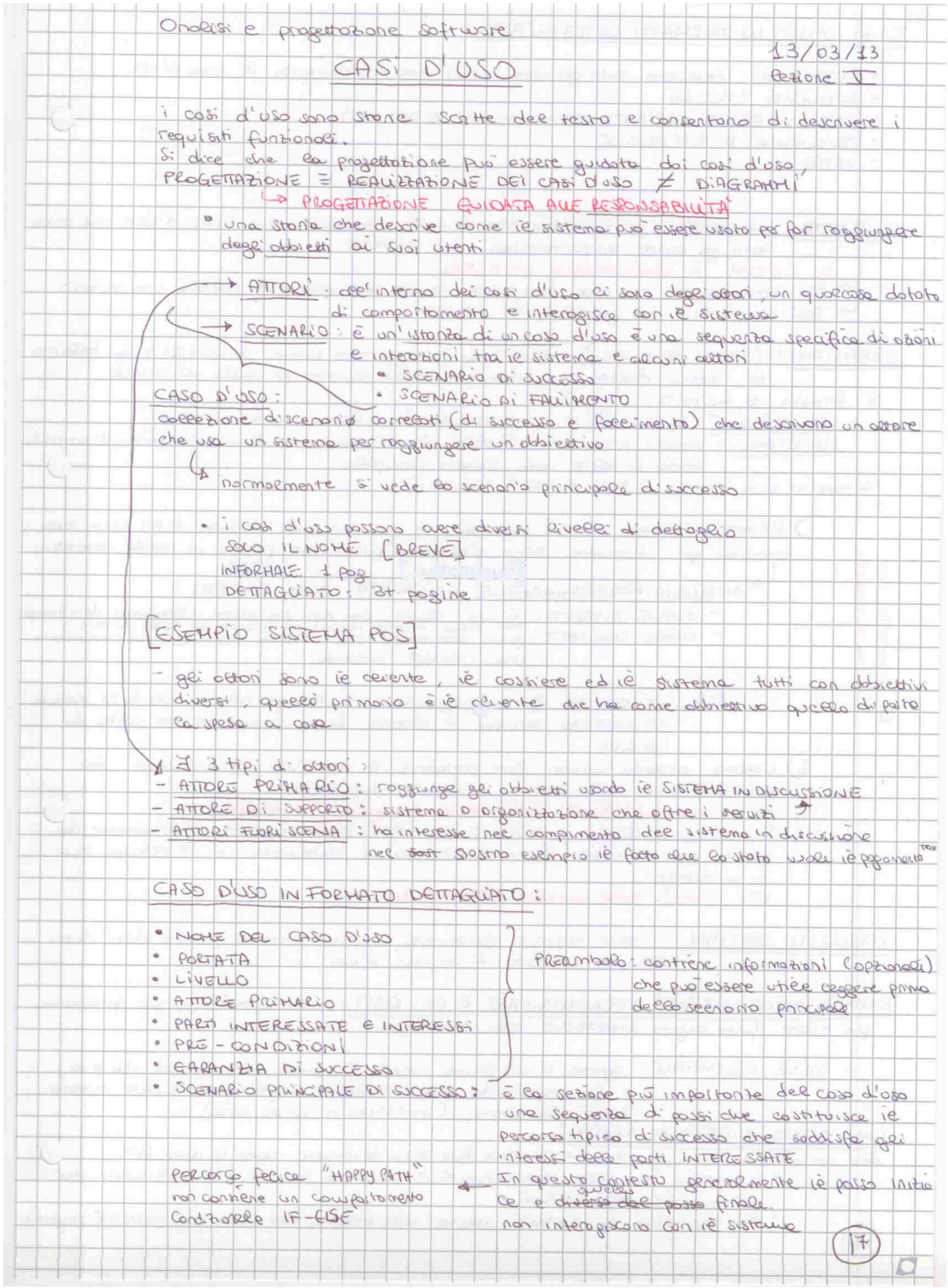
ELABORATI DEI REQUISITI: (sono opzionali)

- Modello dei casi d'uso: un insieme di scenari tipici dell'utilizzo di un sistema usato soprattutto per requisiti funzionali
- Specifiche Supplementari: essenzialmente tutto ciò che non rientra nei casi d'uso → tutti i requisiti non funzionali
- Glossario: definisce nella sua forma più semplice i termini significativi del dominio, ha un ruolo di dizionario dei dati che registra i requisiti relativi ai dati
- Regole di Business: sono richieste nel dominio o nel business ed è possibile che molte applicazioni vi si debbano conformare.
  Es: leggi fiscali dello stato

UN OBBIETTIVO FONDAMENTALE DELLA GESTIONE DEI REQUISITI È STABILIRE UN LINGUAGGIO COMUNE SU CUI POTER BASARE LA COMUNICAZIONE

COME POTER TROVARE I REQUISITI:

→ il cliente tenta di formulare una descrizione ricercata un po' vaga dei dati delle funzioni e del comportamento del sistema
→ lo sviluppatore formula i requisiti in modo efficace per lo sviluppo del software
→ un obbiettivo FONDAMENTALE della gestione dei requisiti è STABILIRE un linguaggio comune accettato delle diverse parti interessate su cui POTER BASARE UNA COMUNICAZIONE EFFICACE DURANTE IL PROGETTO.



Analisi e progettazione software

13/03/13
Lezione V

CASI D'USO

I casi d'uso sono storie scritte dei testo e consentono di descrivere i requisiti funzionali.
Si dice che la progettazione può essere guidata dai casi d'uso,
PROGETTAZIONE ≡ REALIZZAZIONE DEI CASI D'USO ≠ DIAGRAMMI
→ PROGETTAZIONE GUIDATA DALLE RESPONSABILITÀ

- una storia che descrive come il sistema può essere usato per far raggiungere degli obiettivi ai suoi utenti

→ ATTORI: dell'interno dei casi d'uso ci sono degli attori, un qualcosa dotato di comportamento e interagisce con il sistema
→ SCENARIO: è un'istanza di un caso d'uso è una sequenza specifica di azioni e interazioni tra il sistema e alcuni attori
- SCENARIO di successo
- SCENARIO di FALLIMENTO

CASO D'USO: collezione di scenari correlati (di successo e fallimento) che descrivono un attore che usa un sistema per raggiungere un obiettivo

↳ normalmente si vede lo scenario principale di successo

- i casi d'uso possono avere diversi livelli di dettaglio
  SOLO IL NOME [BREVE]
  INFORMALE: 1 pag
  DETTAGLIATO: 3+ pagine

[ESEMPIO SISTEMA POS]

- gli attori sono il cliente, il cassiere ed il sistema tutti con obiettivi diversi, quello primario è il cliente che ha come obiettivo quello di pagare la spesa e casa

↳ I 3 tipi di attori:
- ATTORE PRIMARIO: raggiunge gli obiettivi usando il SISTEMA IN DISCUSSIONE
- ATTORE DI SUPPORTO: sistema o organizzazione che offre i servizi
- ATTORI FUORI SCENA: ha interesse nel compimento del sistema in discussione
  nel ~~sost~~ nostro esempio il fisco che lo stato vuole il pagamento

CASO D'USO INFORMATO DETTAGLIATO:

- NOME DEL CASO D'USO
- PORTATA
- LIVELLO
- ATTORE PRIMARIO
- PARTI INTERESSATE E INTERESSI
- PRE-CONDIZIONI
- GARANZIA DI SUCCESSO
- SCENARIO PRINCIPALE DI SUCCESSO: è la sezione più importante del caso d'uso una sequenza di passi che costituisce il percorso tipico di successo che soddisfa gli interessi delle parti INTERESSATE
  In questo contesto generalmente il passo iniziale e diverso dal passo finale.
  non interagiscono con il sistema

PREAMBOLO: contiene informazioni (opzionali) che può essere utile leggere prima dello scenario principale

PERCORSO FELICE "HAPPY PATH" non contiene un comportamento condizionale IF-ELSE

17

CASO D'USO IN FORMATO DETTAGLIATO

- ESTENSIONI: descrivono tutti gli altri scenari che compongono il caso d'uso
- REQUISITI SPECIALI
- ELENCO DELLE VARIANTI TECNOLOGICHE E DEI DATI
- FREQUENZA DI RIPETIZIONE
- VARIE

IL SISTEMA: concretizza un contratto tra le parti interessate con i casi d'uso che descrivono gli aspetti comportamentali del contratto
- IL SISTEMA È CONSIDERATO UN ATTORE
- PRE-CONDIZIONI: che cosa deve essere sempre vero prima di iniziare uno scenario
- POST-CONDIZIONI: deve garantire il successo

ESTENSIONI: (flussi alternativi) descrivono tutti gli altri scenari di successo o di fallimento. Vanno descritti come deviazioni dallo scenario principale di successo

è formato da due parti:
- CONDIZIONE: qualcosa che può essere ricevuto da un sistema o da un attore
- GESTIONE: singolo o sequenza di passi, dopo l'esecuzione di un'estensione, e controlla torna di solito allo scenario principale

~~ci sono~~ ci sono almeno 3 tipi diversi di estensioni:

1) Scrivere nel ~~passo~~ scenario principale il passo in modo astratto e poi usare delle estensioni per descrivere come quel passo può essere portato in modo concreto
   - ANALOGIA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI:
     - CLASSE ASTRATTA: è una classe che non può essere istanziata direttamente non posso scrivere new() → concetto che esiste di fatto
     - CLASSE CONCRETA: posso istanziare una classe concreta che implementa una classe astratta

   ES: pagamento con carta di credito

   ES: un cliente fa un pagamento → rimango astratto non dico come concretizzo dicendo il cliente fa un pagamento con carta di credito.
2) L'attore primario vuole far procedere il caso d'uso diversamente da quanto previsto nello scenario principale di successo
   ES: ci sono più articoli della stessa categoria di prodotto
3) Si verifica qualcosa per cui il caso d'uso deve procedere diversamente da quanto previsto nello scenario principale → solitamente è il sistema che se ne accorge
   ES: codice dell'articolo NON valido

REQUISITI SPECIALI: sono requisiti funzionali, qualità e vincoli, che è utile descrivere nel contesto di un caso d'uso

ELENCO DELLE VARIANTI TECNOLOGICHE E DEI DATI: diversi modi in cui qualcosa può essere fatto (legato soprattutto alla tecnologia)
- STILE ESSENZIALE: ignora l'interfaccia utente (suono interessanti al caso) e fa riferimento alle intenzioni degli utenti e alla responsabilità del sistema (utilizzato in questo corso)

CASI D'USO A SCATOLA NERA:
descrivono che cosa il sistema deve fare senza descrivere come lo deve fare
- il sistema ha delle responsabilità e collabora con gli attori esterni che hanno delle responsabilità
- non descrivono la struttura interna del sistema nei suoi componenti o il progetto

Analisi e progettazione software

14/03/13
Lezione VI

MODELLO DEI CASI D'USO → requisiti funzionali

UP prevede vetenon elaborati per descrivere altre tipologie di requisiti

↳ SPECIFICHE SUPPLEMENTARI: requisiti non funzionali; descritti male dai casi d'uso
↳ GLOSSARIO: raccoglie termini e definizioni importanti usate nel progetto
↳ VISIONE: descrive il sistema da realizzare dal p.to di vista delle parti interessate
① sintesi dei requisiti fondamentali desiderati
② base per i requisiti tecnici più dettagliati
↳ REGOLE DI DOMINIO: sono regole che impongono dei modi di operare
es: leggi giuridiche, leggi fisiche (regole di business)

ITERAZIONE 1

nella prima iterazione enfasi è posta su disegnare la responsabilità
Il grosso di questa fase preliminare è quello di comprendere i requisiti progettare la scheletro dell'architettura.

N.B: sue libro, nell'ambito dell'iterazione 1 vengono volutamente commessi degli errori di analisi e progettazione
Lo spunto con cui è stata fatta questa scelta è: ① è comune commettere degli errori ② è utile capire che è possibile gestire gli errori commessi

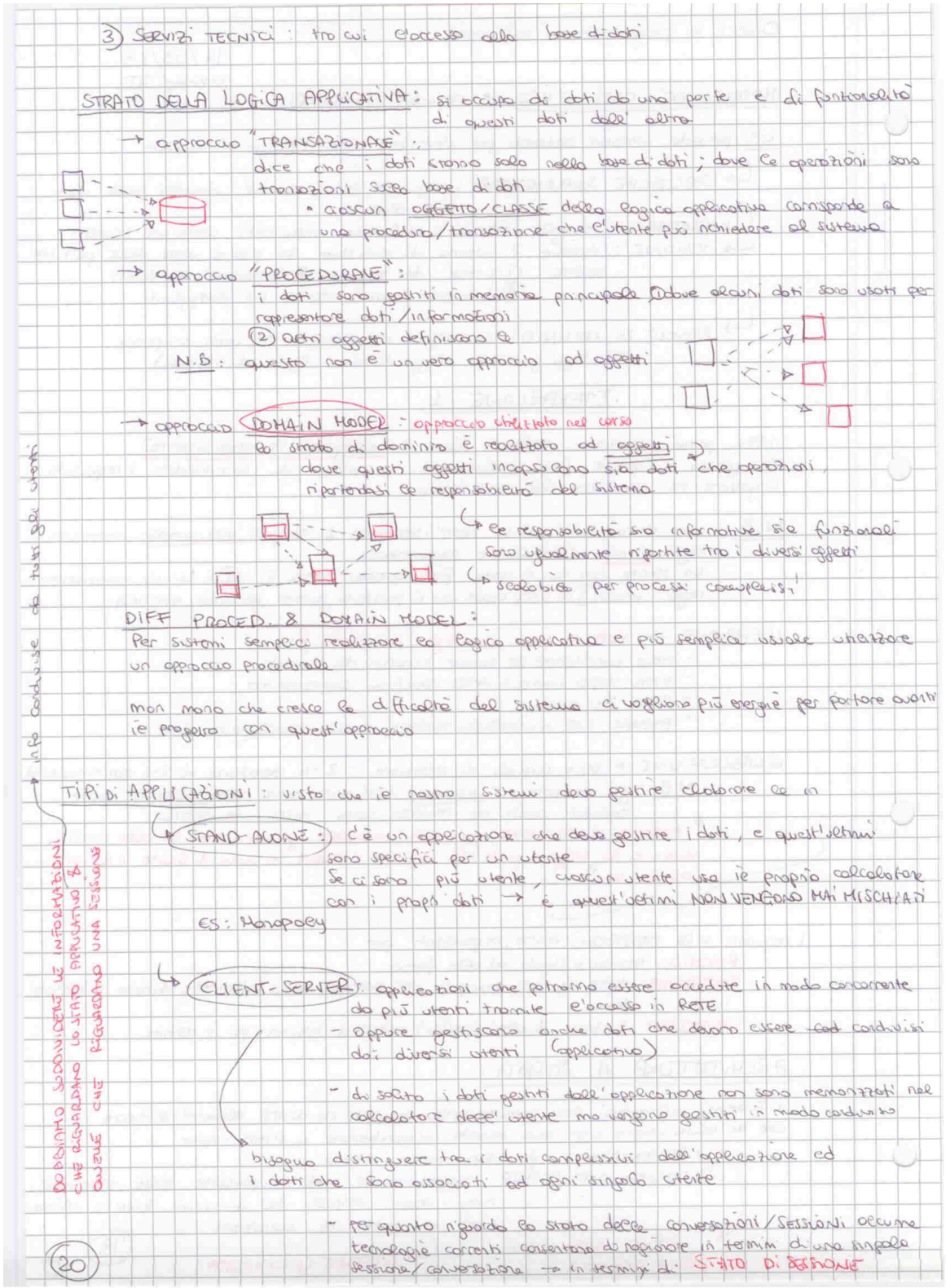
IDEAZIONE → Vale la pena investigare seriamente?
- breve workshop per avviare l'analisi dei requisiti
- bozza della visione e delle specifiche supplementari
- lista dei rischi
- proposta sull'architettura candidata e sulle tecnologie da usare

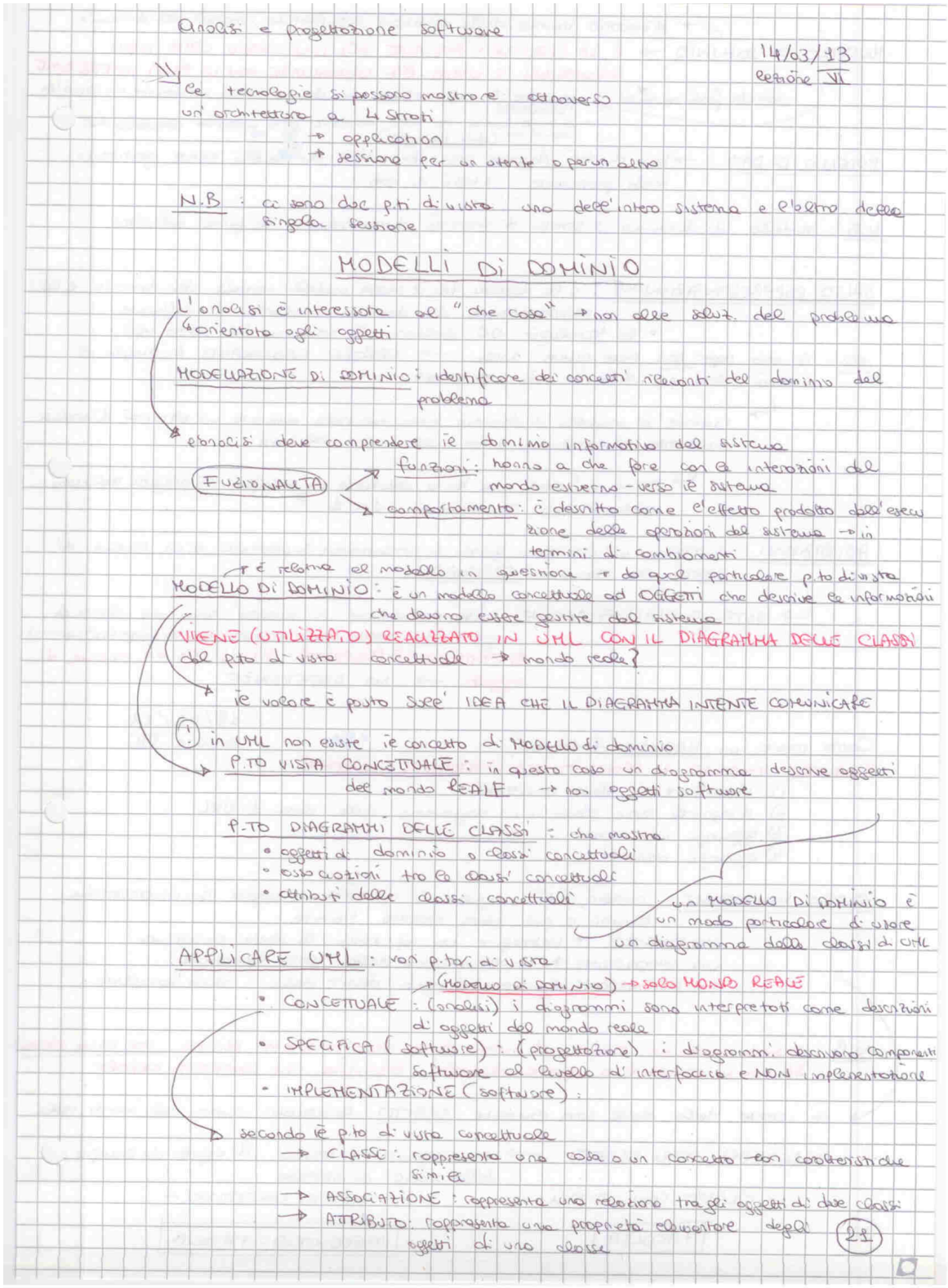
ELABORAZIONE → serie iniziale di iterazioni: 2-4 iterazione di 2-6 sett → timeboxed
N.B: gli elementi più rischiosi dell'architettura ~~VENGONO~~ DEVONO essere realizzati subito per rendersi conto della tempistica
"costruire l'architettura principale, risolvere gli elementi ad alto rischio definire la maggior parte dei requisiti e stimare il piano di lavoro e le risorse complessive"

i requisiti e le ITERAZIONI vanno organizzati per
RISCHIO: tecnico o legato ad altri fattori
COPERTURA: tutte le parti principali del sistema vanno affrontate durante l'elaborazione
CRITICITÀ: funzioni di alto (livello) valore di business per il cliente

ARCHITETTURA A STRATI:

i sistemi informatici sono comunemente composti da diversi elementi software che di solito vengono organizzati secondo un'architettura a strati - ogno
1) PRESENTAZIONE: è l'interfaccia utente
2) LOGICA APPLICATIVA: coppiamento il modo indipendente dalle tecnologie tutti i diversi concetti del dominio, tutte le informazioni e tutte le operazioni
CAPIRE CHE/CHI È LA RICHIESTA → GESTIRE LA RICHIESTA


3) SERVIZI TECNICI: tra cui l'accesso alla base di dati
STRATO DELLA LOGICA APPLICATIVA: si occupa di dati da una parte e di funzionalità di questi dati dall'altra
→ approccio "TRANSAZIONALE":
dice che i dati stanno solo nella base di dati; dove le operazioni sono transazioni sulla base di dati
• ciascun OGGETTO/CLASSE della logica applicativa corrisponde a una procedura/transazione che l'utente può richiedere al sistema
→ approccio "PROCEDURALE":
i dati sono gestiti in memoria principale dove alcuni dati sono usati per rappresentare dati/informazioni
(2) altri oggetti definiscono le
N.B.: questo non è un vero approccio ad oggetti
→ approccio DOMAIN MODEL: approccio studiato nel corso
lo strato di dominio è realizzato ad oggetti
dove questi oggetti incapsulano sia dati che operazioni riportandosi le responsabilità del sistema
le responsabilità sia informative sia funzionali sono ugualmente ripartite tra i diversi oggetti
adeguabile per processi complessi
DIFF PROCED. & DOMAIN MODEL:
Per sistemi semplici realizzare la logica applicativa è più semplice usabile utilizzare un approccio procedurale
man mano che cresce la difficoltà del sistema ci vogliono più energie per portare avanti le progetto con quest'approccio
TIPI di APPLICAZIONI: visto che i nostri sistemi devo gestire elaborare e in
STAND-ALONE: c'è un applicazione che deve gestire i dati, e quest'ultimi sono specifici per un utente
Se ci sono più utente, ciascun utente usa il proprio calcolatore con i propri dati → e quest'ultimi NON VENGONO MAI MISCHIATI
ES: Monopoly
CLIENT-SERVER: applicazioni che possono essere accedute in modo concorrente da più utenti tramite l'accesso in RETE
- Oppure gestiscono anche dati che devono essere condivisi dai diversi utenti (applicativo)
- di solito i dati gestiti dall'applicazione non sono memorizzati nel calcolatore dell'utente ma vengono gestiti in modo condiviso
bisogna distinguere tra i dati complessivi dell'applicazione ed i dati che sono associati ad ogni singolo utente
- per quanto riguarda lo stato delle conversazioni/SESSIONI alcune tecnologie correnti consentono di registrare in termini di una singola sessione/conversazione → in termini di: STATO DI SESSIONE
DOBBIAMO SEPARARE LE INFORMAZIONI CHE RIGUARDANO LO STATO APPLICATIVO DI UNA SESSIONE
20



Analisi e progettazione software

14/03/13
Lezione VI

Ce tecnologie si possono mostrare attraverso un'architettura a 4 strati
→ application
→ sessione per un utente o per un altro

N.B. : ci sono due p.ti di vista uno dell'intero sistema e l'altro della singola sessione

MODELLI DI DOMINIO

L'analisi è interessata al "che cosa" → non alle soluz. del problema orientata agli oggetti

MODELLAZIONE DI DOMINIO: identificare dei concetti rilevanti del dominio del problema

analisi: deve comprendere il dominio informativo del sistema

FUNZIONALITÀ
- funzioni: hanno a che fare con le interazioni del mondo esterno - verso il sistema
- comportamento: è descritto come l'effetto prodotto dall'esecuzione delle operazioni del sistema → in termini di cambiamenti

MODELLO DI DOMINIO: è un modello concettuale ad OGGETTI che descrive le informazioni che devono essere gestite dal sistema
→ è relativo al modello in questione → da quel particolare p.to di vista

VIENE (UTILIZZATO) REALIZZATO IN UML CON IL DIAGRAMMA DELLE CLASSI
dal p.to di vista concettuale → mondo reale?

ie valore è posto sull'IDEA CHE IL DIAGRAMMA INTENDE COMUNICARE

! in UML non esiste il concetto di MODELLO di dominio

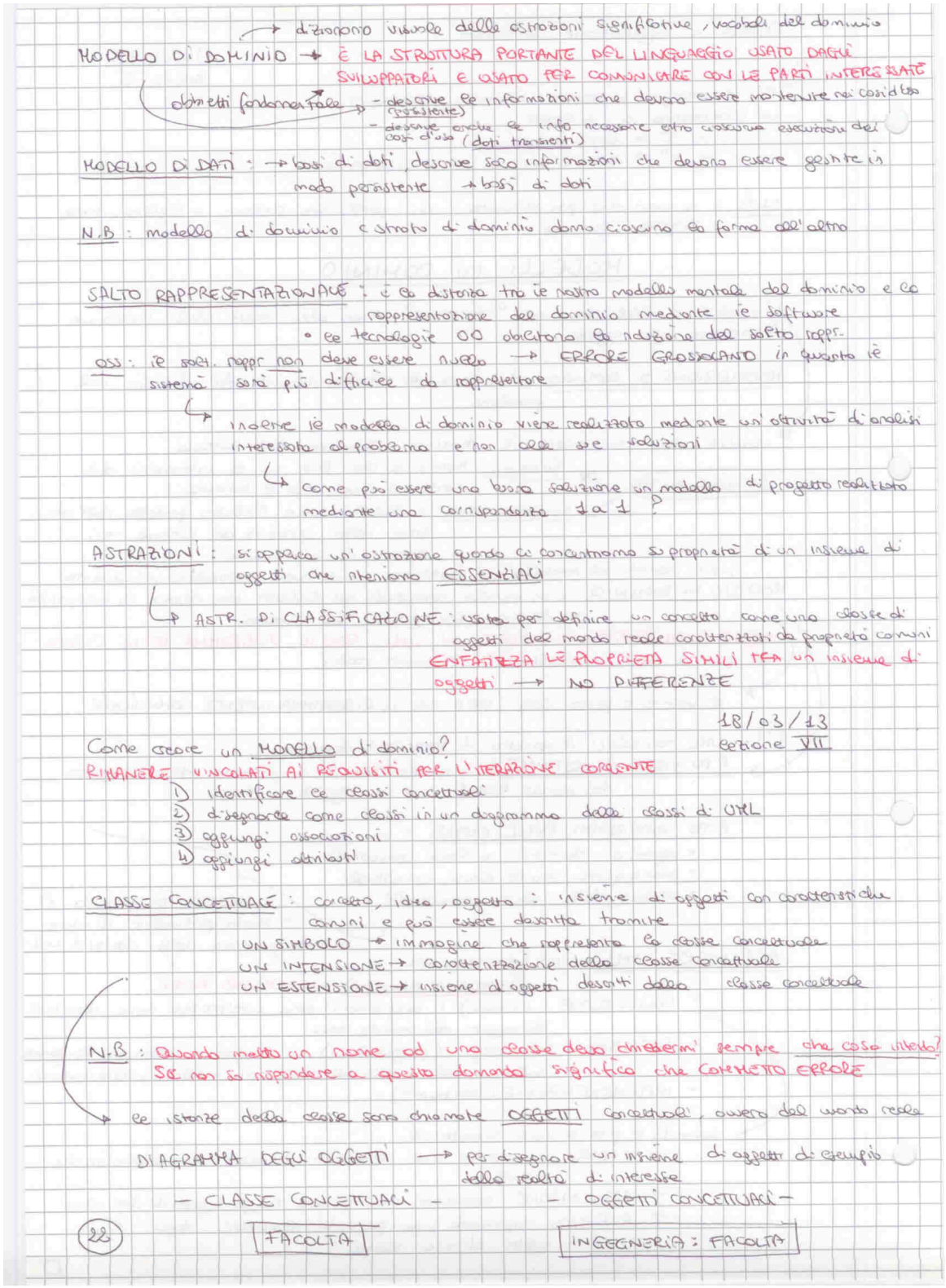
P.TO VISTA CONCETTUALE: in questo caso un diagramma descrive oggetti del mondo REALE → non oggetti software

P.TO DIAGRAMMI DELLE CLASSI: che mostra
- oggetti di dominio o classi concettuali
- associazioni tra le classi concettuali
- attributi delle classi concettuali

un MODELLO DI DOMINIO è un modo particolare di usare un diagramma delle classi di UML

APPLICARE UML: vari p.ti di vista
(Modello di dominio) → solo MONDO REALE
- CONCETTUALE: (analisi) i diagrammi sono interpretati come descrizioni di oggetti del mondo reale
- SPECIFICA (software): (progettazione): diagrammi descrivono componenti software al livello di interfaccia e NON implementazione
- IMPLEMENTAZIONE (software):

secondo il p.to di vista concettuale
→ CLASSE: rappresenta una cosa o un concetto con caratteristiche simili
→ ASSOCIAZIONE: rappresenta una relazione tra gli oggetti di due classi
→ ATTRIBUTO: rappresenta una proprietà ciascuno degli oggetti di una classe


→ dizionario visuale delle astrazioni significative, vocabolo del dominio
MODELLO DI DOMINIO → È LA STRUTTURA PORTANTE DEL LINGUAGGIO USATO DAGLI SVILUPPATORI E USATO PER COMUNICARE CON LE PARTI INTERESSATE
obiettivi fondamentale → - descrive le informazioni che devono essere mantenute nel corso del tempo (persistente)
- descrive anche le info necessarie oltre l'esecuzione del caso d'uso (dati transienti)
MODELLO DI DATI : → basi di dati descrive solo informazioni che devono essere gestite in modo persistente → basi di dati
N.B : modello di dominio e strato di dominio sono ciascuno la forma dell'altro
SALTO RAPPRESENTAZIONALE : è la distanza tra il nostro modello mentale del dominio e la rappresentazione del dominio mediante il software
• le tecnologie OO abilitano la riduzione del salto rappr.
oss: il salto rappr non deve essere troppo → ERRORE GROSSOLANO in quanto il sistema sarà più difficile da rappresentare
↳ inoltre il modello di dominio viene realizzato mediante un'attività di analisi interessata al problema e non alle sue soluzioni
↳ come può essere una buona soluzione un modello di progetto realizzato mediante una corrispondenza 1 a 1 ?
ASTRAZIONI : si applica un'astrazione quando ci concentriamo su proprietà di un insieme di oggetti che riteniamo ESSENZIALI
↳ ASTR. DI CLASSIFICAZIONE : usata per definire un concetto come una classe di oggetti dal mondo reale caratterizzati da proprietà comuni
ENFATIZZA LE PROPRIETÀ SIMILI TRA un insieme di oggetti → NO DIFFERENZE
18/03/13
Lezione VII
Come creare un MODELLO di dominio?
RIMANERE VINCOLATI AI REQUISITI PER L'ITERAZIONE CORRENTE
1) identificare le classi concettuali
2) disegnarle come classi in un diagramma delle classi di UML
3) aggiungi associazioni
4) aggiungi attributi
CLASSE CONCETTUALE : concetto, idea, oggetto : insieme di oggetti con caratteristiche comuni e può essere descritta tramite
UN SIMBOLO → IMMAGINE che rappresenta la classe concettuale
UN INTENSIONE → caratterizzazione della classe concettuale
UN ESTENSIONE → insieme di oggetti descritti dalla classe concettuale
N.B : Quando metto un nome ad una classe devo chiedermi sempre che cosa intendo? Se non so rispondere a questa domanda significa che COMMETTO ERRORE
le istanze della classe sono chiamate OGGETTI concettuali ovvero del mondo reale
DIAGRAMMA DEGLI OGGETTI → per disegnare un insieme di oggetti di esempio della realtà di interesse
- CLASSE CONCETTUALI - - OGGETTI CONCETTUALI -
FACOLTÀ
INGEGNERIA : FACOLTÀ
22

Analisi e progettazione software

18/03/13
Lezione III

NOTAZIONE: - se ci sono i due punti è un oggetto
- se non ci sono è una CLASSE

ASSOCIAZIONE: è una relazione tra le classi → istanza di queste classi
↳ ciascuna istanza di un'associazione è chiamata COLLEGAMENTO
i collegamenti sono bidirezionali

ATTRIBUTO: è una proprietà elementare un dato degli oggetti di una classe
↳ è di solito si tratta di un tipo primitivo
↳ funzioni che associano a ciascun oggetto un VALORE

Diagramma di oggetti di dominio → modo visuale di rappresentare un insieme di oggetti del mondo reale: attributo - valore

STRATEGIE IDENTIFICAZIONI CLASSI CONCETTUALI
→ Quali sono le linee guida per identificare le classi concettuali?

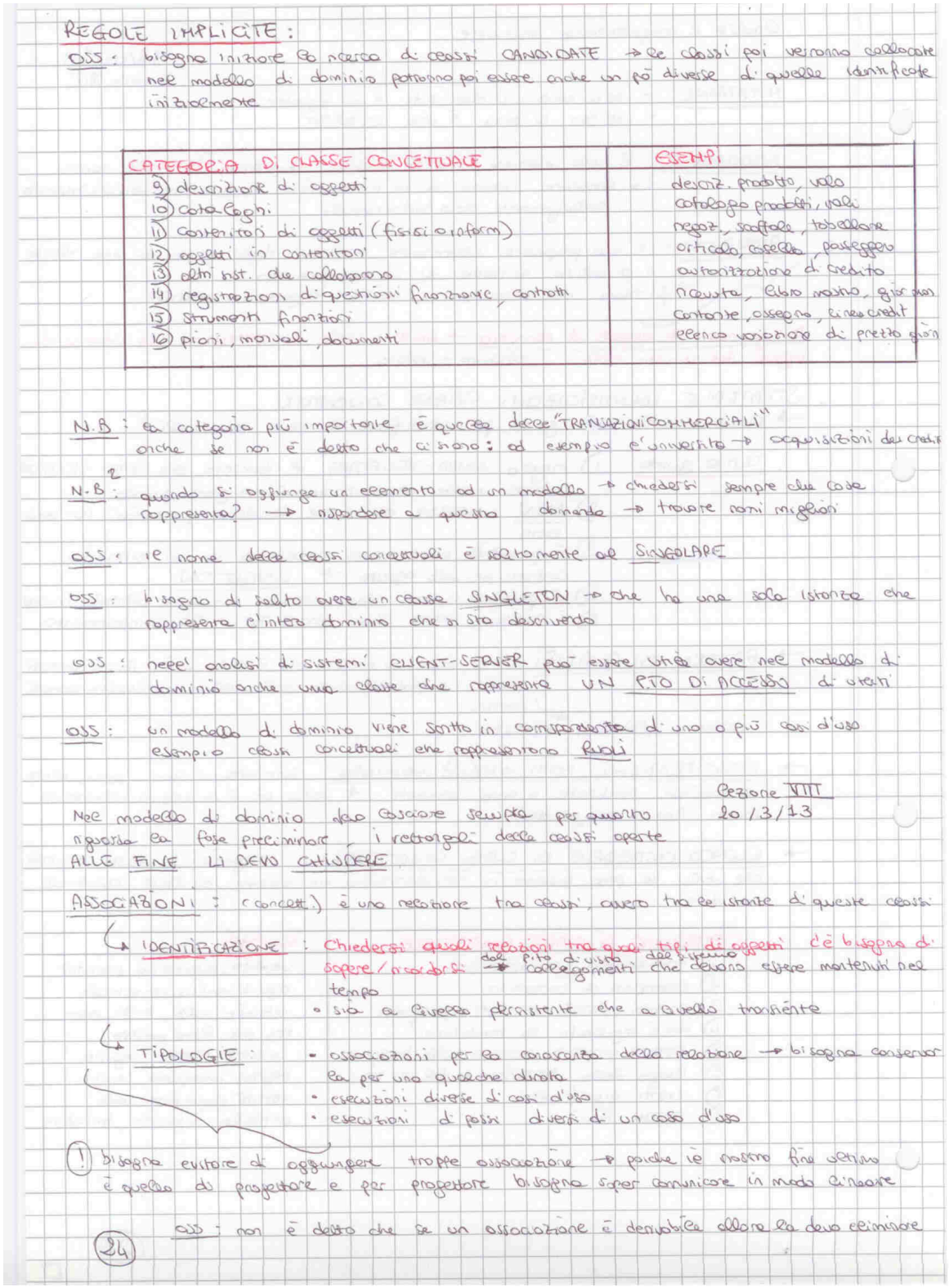
Linee guida
1) meglio SOVRA-SPECIFICARE il dominio che SOTTO-SPECIFICARE ovvero molte classi concettuali a grana piccola
2) NON escludere classi che descrivono informazioni non persistenti
3) è possibile avere classi concettuali la cui estensione contiene un solo oggetto → SINGLETON
4) è possibile avere classi concett. con senza attributi (control)
5) è possibile avere classi concett. con un solo comportamento

→ PATTERN DI ANALISI: un'idea di analisi che è stata utile in più contesti pratici e probabilmente sarà utile anche in altri contesti pratici
risolvono problemi specifici

→ IDENTIFICAZIONI NOMI → LOGOS. NOMINALI: considera i nomi come classi concettuali candidate o come attributi → anche se è un elenco abbastanza povero (poco consigliato)

→ ELENCO CATEGORIE DI CLASSI CONCETTUALI: categoria di classi concettuali che vale la pena prendere in considerazione un insieme di classi concettuali candidate.

CATEGORIE DI CLASSE CONCETTUALE	ESEMPI
1) transazioni commerciali	vendita, pagamento, prenotazione
2) elementi di transazioni	riga di vendita per articolo
3) prodotti o servizi correlati a transazioni	articolo, volo, posto
4) dove è registrata la transazione?	registro, libro mastro
5) persone/organizz. correlati nelle transazione	cassiere, cliente, negozio
6) luogo della transazione o del servizio	negozio, aeroporto, posto
7) eventi significativi	vendita, pagamento, volo
8) oggetti fisici	articolo, registro, tabellone



REGOLE IMPLICITE:

OSS: bisogna iniziare la ricerca di classi CANDIDATE → le classi poi verranno collocate nel modello di dominio potranno poi essere anche un pò diverse di quelle identificate inizialmente

CATEGORIA DI CLASSE CONCETTUALE	ESEMPI
9) descrizione di oggetti	descriz. prodotto, volo
10) catalogo	catalogo prodotti, voli
11) contenitori di oggetti (fisici o informa)	negoz., scaffale, tabellone orticolo, cassetta, passeggero autorizzazione di credito
12) oggetti in contenitori	rewrite, libro mastro, giornale contabile, assegno, linea credit elenco versazione di pretto giorn
13) altri sist. che collaborano	
14) registrazioni di questioni finanziarie, contratti	
15) strumenti finanziari	
16) piani, manuali, documenti	

N.B: la categoria più importante è quella delle "TRANSAZIONI COMMERCIALI" anche se non è detto che ci siano: ad esempio l'università → acquisizioni dei crediti

N.B 2: quando si aggiunge un elemento ad un modello → chiedersi sempre che cosa rappresenta? → rispondere a questa domanda → trovare nomi migliori

OSS: le nome delle classi concettuali è solitamente al SINGOLARE

OSS: bisogna di solito avere un classe SINGLETON → che ha una sola istanza che rappresenta l'intero dominio che si sta descrivendo

OSS: nelle analisi di sistemi CLIENT-SERVER può essere utile avere nel modello di dominio anche una classe che rappresenta UN PUNTO DI ACCESSO di utenti

OSS: un modello di dominio viene scritto in corrispondenza di uno o più casi d'uso esempio classi concettuali che rappresentano RUOLI

Lezione VIII
20/3/13

Nel modello di dominio deve esistere semplicemente per questo motivo la fase preliminare, i rettangoli delle classi aperte ALLE FINE LI DEVO CHIUDERE

ASSOCIAZIONI: (concett.) è una relazione tra classi, quindi tra le istanze di queste classi:

↳ IDENTIFICAZIONE: Chiedersi quali relazioni tra quali tipi di oggetti è bisogna di sapere / ricordarsi dal punto di vista del sistema → collegamenti che devono essere mantenuti nel tempo
- sia a livello persistente che a quello transiente

↳ TIPOLOGIE:
- associazioni per la comodità della relazione → bisogna conservar la per una qualche durata
- esecuzioni diverse di casi d'uso
- esecuzioni di passi diversi di un caso d'uso

(!) bisogna evitare di aggiungere troppe associazione → perché le nostre fine ultimo è quello di progettare e per progettare bisogna saper comunicare in modo chiaro

OSS: non è detto che se un associazione è derivabile allora la devo eliminare

Analisi e progettazione software

Lezione VIII
20/3/13

MODELLO DI PROGETTO ≡ MODELLO SOFTWARE

NB: nei modelli software le associazioni sono unidirezionali

NOMI PER LE ASSOCIAZIONI: SONO VERBO / LOCUZIONE VERBALE

- vanno evitati verbi poco informativi es: "ha", "usa"
- convenzioni → MAIUSCOLO e lettura da sinix-dex ; alto-basso

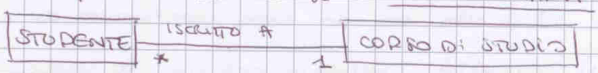
RUOLI: ad ogni estremità di un associazione c'è un RUOLO

↳ la molteplicità di un ruolo indica quante istanze di una classe possono essere associate ad un'istanza dell'altra classe

OSS: la collocazione della molteplicità è inverita rispetto al modello E-R studiato a basi di dati

N.B le molteplicità massima sono PIU IMPORTANTI

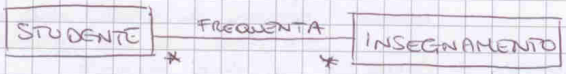
ES:



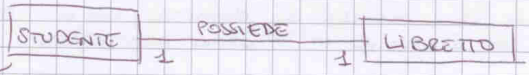
MOLTI A 1

Preso uno studente questo si può iscrivere ad un solo corso di studio

Una molteplicità può dipendere dal dominio oppure dalla prospettiva



MOLTI A MOLTI



1 A 1

ASSOCIAZIONI RIFLESSIVE (RICORSIVE)

① una tecnica per l'identificazione di associazioni concettuali è basata sull'uso di un elenco di categorie comuni

ELENCO ASSOCIAZIONI COMUNI:

CATEGORIA DI ASSOCIAZIONE	ESEMPI
- A è una transazione correlata a un'altra transazione B	pagamento in contanti - vendita prenotazione - cancellazione
- A è un elemento/riga di una transazione B	riga di vendita per l'articolo - vendita
- A è un prodotto o servizio per una transaz. di B	riga di vendita per articolo volo - prenotazione
- A è un ruolo relativo ad una transazione B	cliente - pagamento / passeggero / biglietto
- A è una parte fisica o logica di B	cassetto registratore di cassa
- A è contenuto fisicamente o logicamente in B	casella tabellone / scaffale - aeroplano registratore di cassa - negozio
- A è una descrizione per B	prodotto - scaffale / casella - tabellone descrizione prodotto - articolo descrizione volo - volo
- A è noto/registrato/memorizzato/riportato acquisito in B	vendita - registratore di cassa prenotazione - manifesto volo

nel sistema quando lo considero deve mettere sempre un punto di accesso al sistema.
→ I dati vengono introdotti con gli attributi

CLASSI DI DESCRIZIONE: contengono informazioni che descrivono qualcos'altro
↳ classi le cui istanze non sono oggetti reali

- UTILIZZO UNA CLASSE DI DESCRIZIONE -

↳ quando è necessaria una descrizione di un articolo o servizio, indipendentemente dall'attuale esistenza di istanze di tali articolo/i o servizi

↳ per gestire ANOMALIE ad esempio anomalie di inserimento, eliminazione
ES: sconto preannunciato per un determinato prodotto

(!) perché è meglio sovra-specificare che sotto-specificare

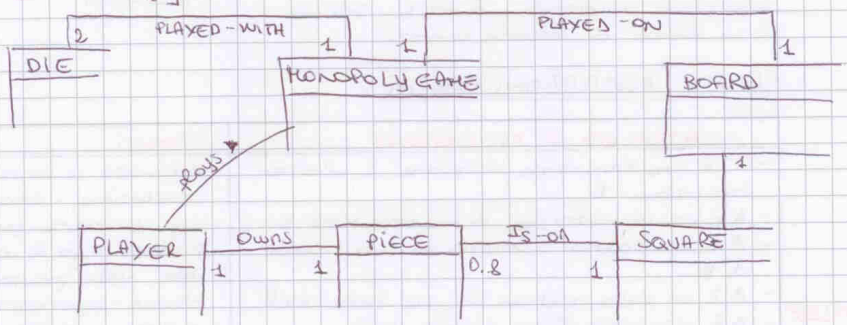
N.B: in UML una classe rappresenta una classe di oggetti ma non è relativo ingrie me → questo (in alcuni casi) va rappresentato con un associazione tra questa classe ed un' altra classe
→ in altri casi può essere rappresentato da un'ulteriore classe concettuale

ASSOCIAZIONI DERIVATA: calcolato mediante la composizione di altre associazioni
→ SINTASSI: il nome va preceduto da /
→ la molteplicità di un associazione derivata può essere solamente derivata da quelle delle associazioni di cui è formata

nel MODELLO di dominio la devo mettere? (perché è un)
- si se nel modello di dominio ha un nome significativo che non è presente in nessun altro associazione

Si mettono quando ho a che fare con relazioni che non possono essere dedotte in modo ovvio dalle altre associazioni del modello di dominio

ES: [MONOPOLY]



Che cosa elenco dalla composizione delle altre associazioni?
↳ sostanzialmente in questo caso nessuna associazione è derivata
CICLO NON SIGNIFICA CHE C'È UN

Analisi e progettazione Software

Lezione VIII
20/03/13

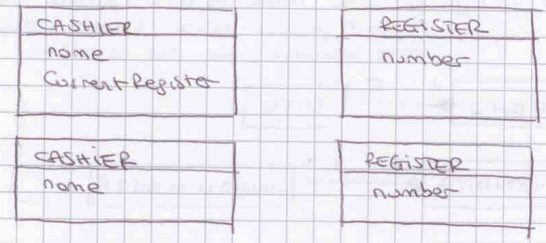
ATTRIBUTO: è una proprietà elementare degli oggetti di una classe, dove a ciascun oggetto della classe associa un valore

↳ Quali sono le linee guida per identificare gli attributi?
↳ deve rispondere alla MASSIMA SEMPLICITÀ

→ ATTRIBUTI DERIVATI: ci metto quando mi serve eseguirlo che è derivato e quindi lo scrivo anche nella gestione
Oppure se ha un nome SIGNIFICATIVO

(!) ERRORE ATTRIBUTI: è un errore mettere un attributo per specificare una chiave esterna

ES:



GLI ATTRIBUTI NON DEVONO ESSERE USATI PER CORRELARE OGGETTI/CLASSI CONCETTUALI NEL MODELLO DI DOMINIO

un altro errore grave è quello di avere attributi i cui valori rappresentano "relazioni" tra classi concettuali nel modello di dominio

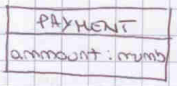
↳ non introdurre DESIGN CREEP = inclusione spesso prematura ed inopportuna di scelte di progetto.

TIPI DI DATI: è un insieme di valori in cui l'identità univoca non è significativa

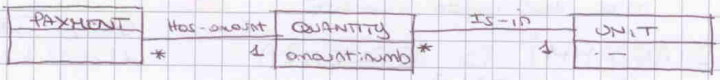
→ In caso di dubbio, definisci una nuova classe concettuale e una associazione

↳ (!) Meglio mettere un tipo di dato ed elaborare gli attributi

QUANTITÀ: sono spesso rappresentate da classi → tipo di dato



→ NON CORRETTO



CORRETTO

IMPORTANZA DEI DIAGRAMMI DEGLI OGGETTI: quest'ultimi sono uno strumento utile per validare e scoprire porzioni del modello di dominio

- si inizi sempre a definire un diagramma degli oggetti che rappresenta una qualche porzione di esempio del dominio
  - ogni oggetto deve essere istanza di una classe concettuale e deve avere un valore per ciascuno degli attributi di quella classe
  - ogni collegamento deve essere istanza di un'associazione concettuale deve collegare una coppia di oggetti delle classi collegate da quell'associazione, i collegamenti devono rispettare le molteplicità delle associazioni

I MODELLI DI DOMINIO NON SONO GIUSTI O SBAGLIATI → la vera domanda da farsi è → IL MODELLO DI DOMINIO È UTILE?

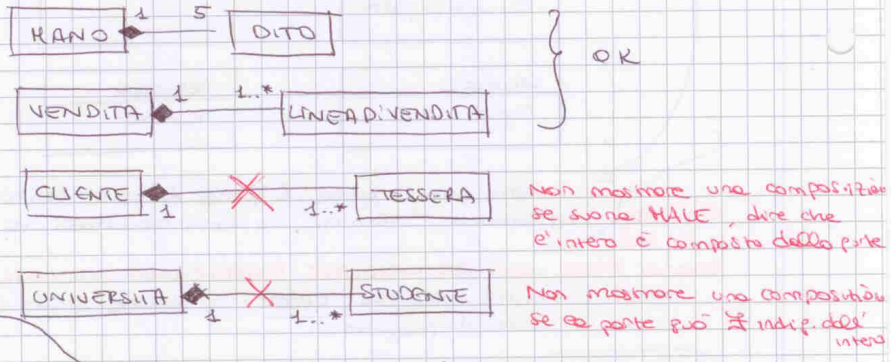
AGGREGAZIONE : è un'associazione che suggerisce una relazione INTERO-PARTE oppure una forte di associazione in cui un oggetto aggregato è composto da parti

- è distinguibile da un'associazione normale se:
  - per descriverla uso la frase è "parte di" o "composto da"?

COMPOSIZIONE : è una forma forte di aggregazione intero-parte in cui
- ciascuna parte deve sempre appartenere a un composto
- il composto è responsabile della creazione e eliminazione delle sue parti

[IN UML] : la vita delle parti è limitata da quella del composto, le parti possono essere create dopo il composto e possono essere distrutte prima del composto.

ES:



in un MODELLO SOFTWARE la composizione può essere usata diversamente dalla composizione in un modello concettuale

ES: a liv. concettuale un università non è composta da studenti
→ tuttavia a livello SOFTWARE potrebbe essere creata identificare una composizione software tra università e studenti

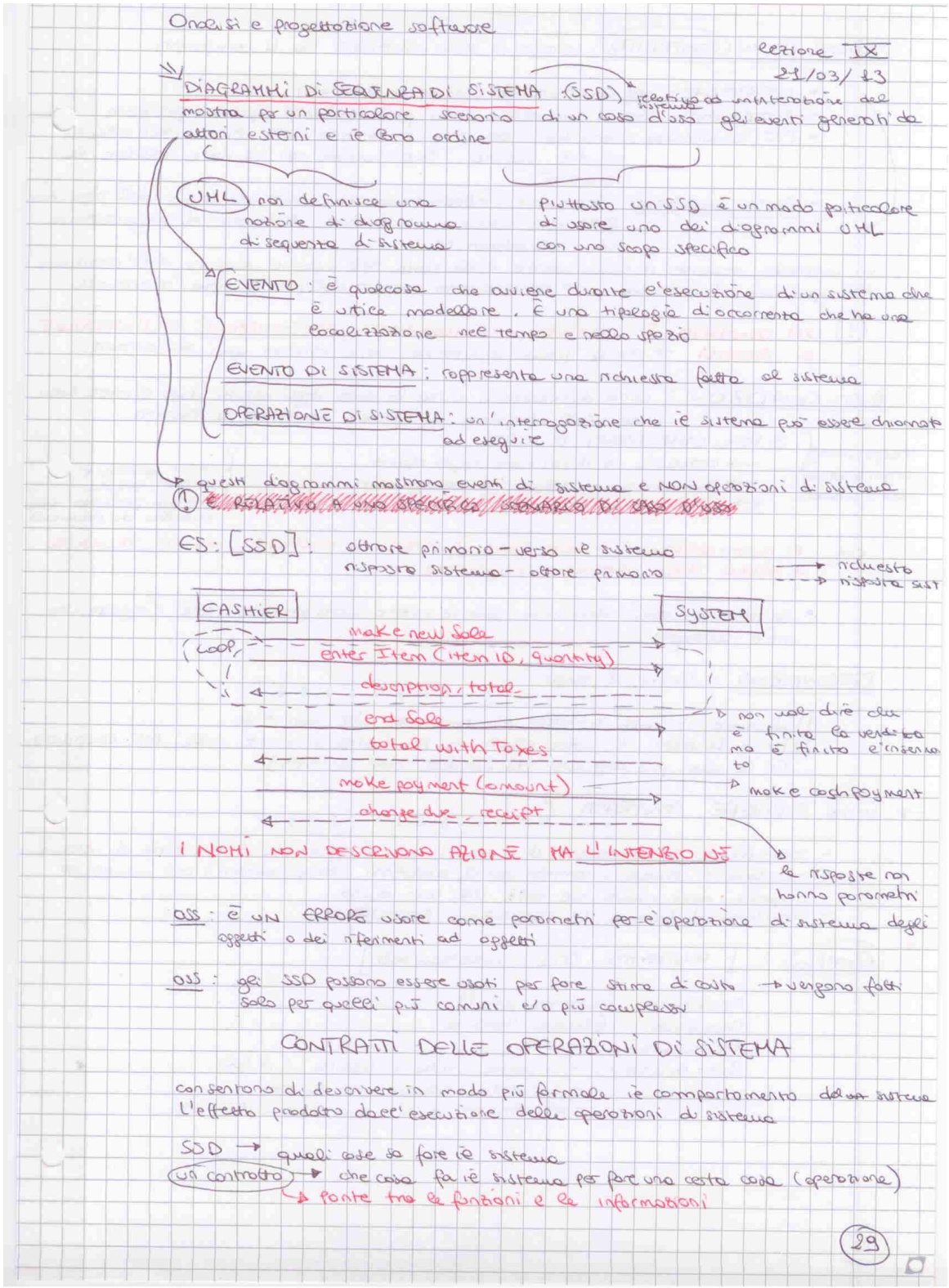
un oggetto studente (SW) verrà probabilmente creato come oggetto soft ware strettamente associato a un oggetto università.
Inoltre la "vita" di un oggetto studente è certamente contenuta nella vita dell'oggetto università.

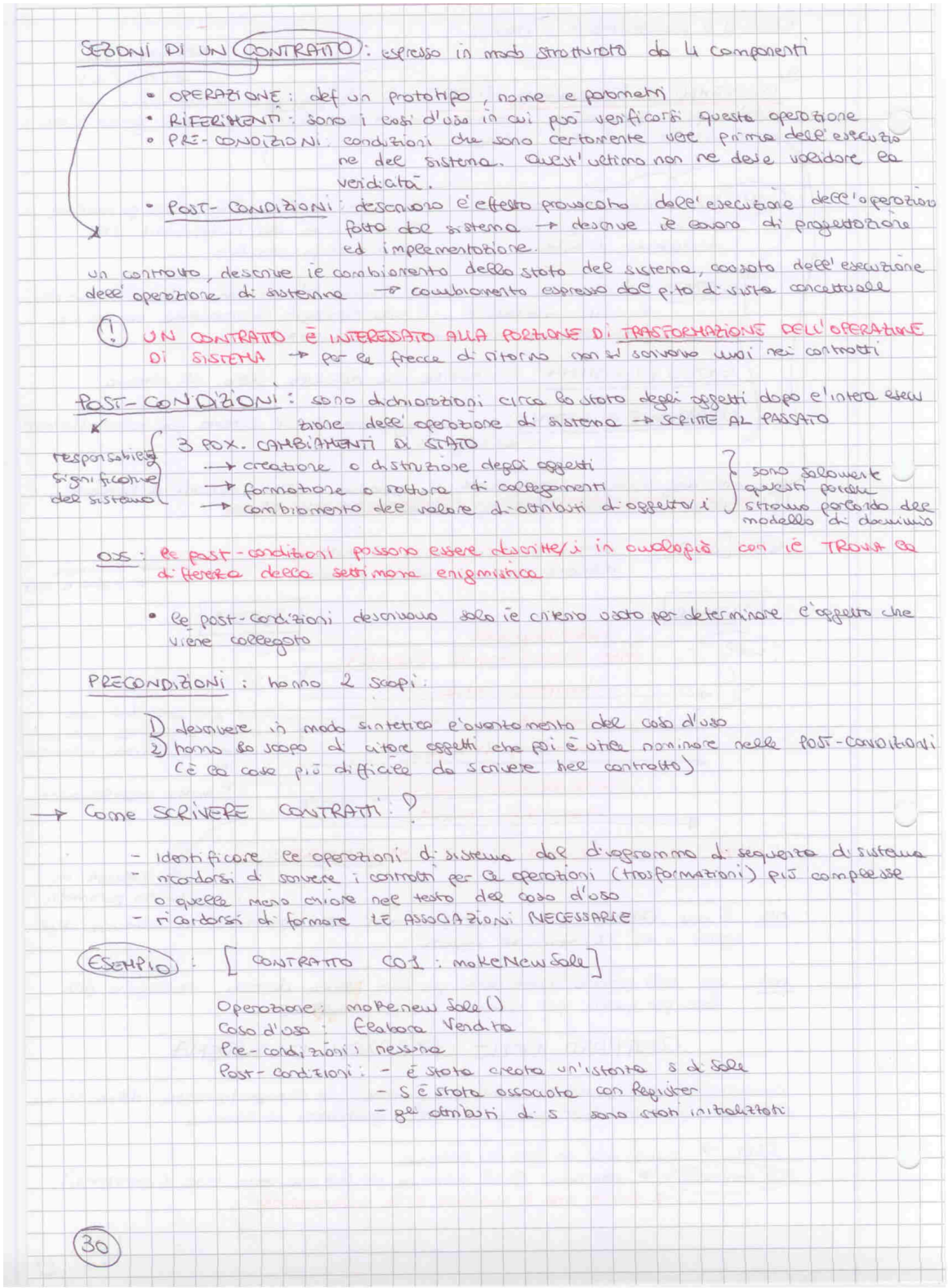
Lezione IX
21/03/13

DIAGRAMMI DI SEQUENZA DI SISTEMA

COMPORTAMENTO : di un sistema viene espresso mediante ① casi d'uso ② diagrammi di sequenza di sistema ③ contratti delle operazioni del sistema.

② DIAGRAMMI DI SEQUENZA DI SISTEMA : descrive che cosa fa fare il sistema in successione ovvero le sue funzioni
- attori che interagiscono con il sistema
- interazioni (esterne) con altri sistemi


Analisi e progettazione software
Lezione IX
21/03/13
DIAGRAMMI DI SEQUENZA DI SISTEMA (SSD)
mostra per un particolare scenario di un caso d'uso gli eventi generati da attori esterni e il loro ordine
UML non definisce una notazione di diagramma di sequenza di sistema
piuttosto un SSD è un modo particolare di usare uno dei diagrammi UML con uno scopo specifico
EVENTO: è qualcosa che avviene durante l'esecuzione di un sistema che è utile modellare. È una tipologia di occorrenza che ha una localizzazione nel tempo e nello spazio
EVENTO DI SISTEMA: rappresenta una richiesta fatta al sistema
OPERAZIONE DI SISTEMA: un'interrogazione che il sistema può essere chiamato ad eseguire
questi diagrammi mostrano eventi di sistema e NON operazioni di sistema
ES: [SSD]: attore primario - verso il sistema
risposta sistema - attore primario
richiesto
risposta sist
CASHIER
SYSTEM
make new Sale
enter Item (item ID, quantity)
description, total
end Sale
total with Taxes
make payment (amount)
change due, receipt
non vuol dire che è finito la vendita ma è finito l'inserimento
make cash payment
I NOMI NON DESCRIVONO AZIONI MA L'INTENTO
le risposte non hanno parametri
OSS: È UN ERRORE usare come parametri per l'operazione di sistema degli oggetti o dei riferimenti ad oggetti
OSS: gli SSD possono essere usati per fare stime di costo → vengono fatti solo per quelli più comuni e/o più complessi
CONTRATTI DELLE OPERAZIONI DI SISTEMA
consentono di descrivere in modo più formale il comportamento del sistema
L'effetto prodotto dall'esecuzione delle operazioni di sistema
SSD → quali cose fa fare il sistema
un contratto → che cosa fa il sistema per fare una certa cosa (operazione)
ponte tra le funzioni e le informazioni
29



SEZIONI DI UN CONTRATTO: espresso in modo strutturato da 4 componenti

- OPERAZIONE: def un prototipo, nome e parametri
- RIFERIMENTI: sono i casi d'uso in cui può verificarsi questa operazione
- PRE-CONDIZIONI: condizioni che sono certamente vere prima dell'esecuzione del sistema. Quest'ultimo non ne deve validare la veridicità.
- POST-CONDIZIONI: descrivono gli effetti provocati dall'esecuzione dell'operazione fatta dal sistema → descrive il lavoro di progettazione ed implementazione.

un contratto, descrive il cambiamento dello stato del sistema, causato dall'esecuzione dell'operazione di sistema → cambiamento espresso dal p.to di vista concettuale

(!) UN CONTRATTO È INTERESSATO ALLA PORZIONE DI TRASFORMAZIONE DELL'OPERAZIONE DI SISTEMA → per le frecce di ritorno non si scrivono mai nei contratti

POST-CONDIZIONI: sono dichiarazioni circa lo stato degli oggetti dopo l'intera esecuzione dell'operazione di sistema → SCRITTE AL PASSATO

responsabilità significative del sistema:
3 POS. CAMBIAMENTI DI STATO
→ creazione o distruzione degli oggetti
→ formazione o rottura di collegamenti
→ cambiamento dei valori di attributi di oggetti
} sono solamente questi perché stiamo parlando del modello di dominio

OSS: le post-condizioni possono essere descritte/i in analogia con le TRUE di tabella della settimana enigmistica

- le post-condizioni descrivono solo il criterio usato per determinare l'oggetto che viene collegato

PRECONDIZIONI: hanno 2 scopi:

1) descrivere in modo sintetico l'avanzamento del caso d'uso
2) hanno lo scopo di usare oggetti che poi è utile nominare nelle POST-CONDIZIONI (è la cosa più difficile da scrivere nel contratto)

→ Come SCRIVERE CONTRATTI?

- identificare le operazioni di sistema dal diagramma di sequenza di sistema
- ricordarsi di scrivere i contratti per le operazioni (trasformazioni) più complesse o quelle meno chiare nel testo del caso d'uso
- ricordarsi di formare LE ASSOCIAZIONI NECESSARIE

ESEMPIO: [CONTRATTO CO1: makeNewSale]

Operazione: makeNewSale()
Caso d'uso: Elabora Vendita
Pre-condizioni: nessuna
Post-condizioni: - è stata creata un'istanza s di Sale
- s è stata associata con Register
- gli attributi di s sono stati inizializzati

Esercizi

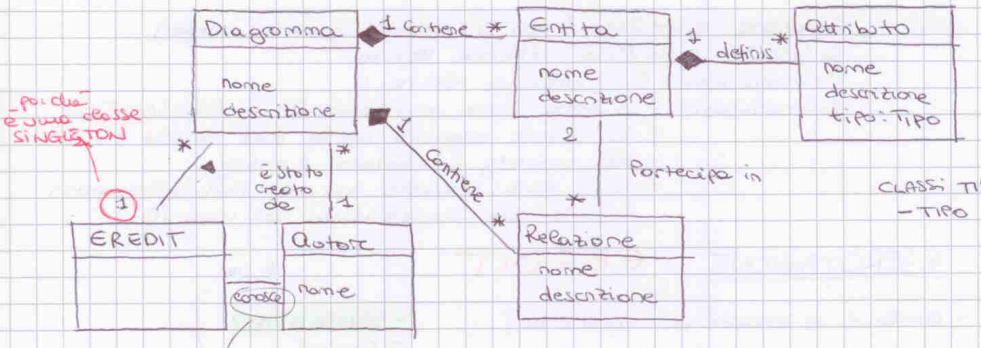
- Fare l'analisi orientata agli oggetti relativa al sistema ERedit, relativamente alle operazioni di sistema identificate, come segue: (1) mostrare il modello di dominio (con riferimento a tutte le operazioni di sistema che sono state identificate); (2) mostrare il contratto delle operazioni di ciascuna delle operazioni di sistema identificate.
- Fare la progettazione orientata agli oggetti relativa al sistema ERedit, come segue: (1) mostrare i diagrammi di interazione relativi alle operazioni di sistema che sono state identificate, motivando le scelte di progetto fatte mediante l'indicazione dei pattern GRASP e GoF applicati; (2) mostrare il corrispondente diagramma delle classi di progetto. Si faccia l'ipotesi che il sistema ERedit gestisca i propri dati solo in memoria principale. Le soluzioni individuate dovranno essere compatibili (in particolare in termini di visibilità, ovvero di navigabilità delle associazioni) con le realizzazioni di tutte le operazioni di sistema e tutte le interrogazioni mostrate.

Estensioni dei requisiti

I seguenti requisiti vanno normalmente considerati uno alla volta e separatamente dagli altri, a meno che non sia richiesto esplicitamente di considerarne più di uno.

A. Requisito: Ad una relazione possono essere associati anche degli attributi, ciascuno con un nome (che lo identifica nell'ambito della relazione), un tipo ed una descrizione.
B. Operazione di sistema (richiede anche requisito A): Inserimento di un nuovo attributo in una relazione del diagramma corrente, dato nome, tipo e descrizione dell'attributo, nonché il nome della relazione a cui l'attributo va associato.
C. Requisito: Un diagramma ER può contenere delle relazioni N-arie, che coinvolgono due o più entità. Ciascuna relazione ha un nome (che la identifica nel diagramma) ed una descrizione.
D. Operazione di sistema (richiede requisito C): Inserimento di una nuova relazione N-aria nel diagramma corrente, dato il nome e la descrizione della relazione, nonché la lista dei nomi delle entità coinvolte dalla relazione.
E. Requisito: Alle relazioni sono associate anche le cardinalità. Per default, quando viene creata una nuova relazione, le cardinalità per tutte le entità coinvolte dalla relazione sono inizialmente uguali a (0,N).
F. Operazione di sistema (richiede requisito E): Modifica delle cardinalità associate ad una partecipazione di una entità in una relazione, dato il nome della relazione, dell'entità e il nuovo valore per le cardinalità.
G. Requisito (requisito C + requisito E): Relazioni N-arie con cardinalità.
H. Operazione di sistema (richiede requisito G): Inserimento di una nuova relazione N-aria nel diagramma corrente, dato il nome e la descrizione della relazione, nonché la lista dei nomi delle entità coinvolte dalla relazione. Le cardinalità per tutte le entità coinvolte dalla relazione sono inizialmente uguali a (0,N).

MODELLO DI DOMINIO :



associazione derivata: la devo togliere? visto che c'è un ciclo la devo togliere? } no perché la composizione di una composizione, può essere ottenuto dalle altre due. NON PUÒ ESSERE DERIVATA.

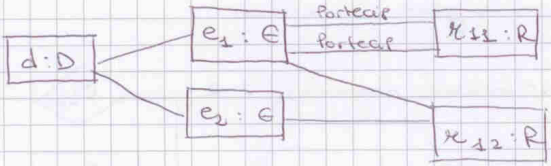
OSS: il grafo degli oggetti di dominio deve essere SEMPRE CONNESSO

ROMBO PIENO → COMPOSIZIONE

ROMBO VUOTO → AGGREGAZIONE SEMPLICE

OSS: la classe singleton va in genere collegata a tutto (DIAGRAMMA)

MODELLO DEGLI OGGETTI :



- Ci sono autori che non hanno ancora creato un diagramma. → ecco perché collego autore - ad EREDIT
- Quali sono le informazioni persistenti che il sistema deve ricordare? ESATTAMENTE IL MODELLO DI DOMINIO

Analisi e progettazione software

Lezione IX
21/03/13

ESEMPIO : [CONTRATTO CO2 : enterITEM]

Operazione : enterItem (itemID : itemID, quantity : integer)
Riferimenti : caso d'uso : Effettua Vendita
Pre-condizioni : È in corso una vendita S
Post-condizioni : - è stata creata un istanza SLI di SaleLineItem
- sli è stata associata con la sale corrente S
- sli.quantity è diventato quantity
- sli è stata associata con una ProductDescription, in base alla corrispondenza con itemID

ESERCITAZIONE : EREDIT

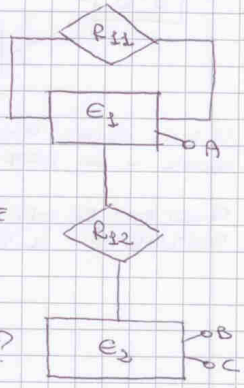
- Quale è il concetto più importante?
- Ci sono ruoli di persone associati a questi concetti?
- Questi concetti dove sono registrati questi concetti?

CONCETTI CANDIDATI:
- DIAGRAMMI
- ENTITÀ
- RELAZIONI
- ATTRIBUTI
- AUTORE
- EREDIT

- Il sistema che sto rappresentando è STAND-ALONE oppure CLIENT-SERVER

OSS: se è stand-alone basta introdurre una classe che rappresenta il sistema invece se è CLIENT-SERVER, si dovrebbe trovare una classe che fornisca l'accesso dell'attore del sistema

Andiamo a disegnare un possibile diagramma E-R



OSS: se ci fossero due entità con lo stesso nome? un entità appartiene ad un solo diagramma

Posso pensare ad entità senza disegnarla? No è visto che è contenuta in è UNA COMPOSIZIONE

OSS: Partecipazione potrebbe essere una classe? in questo caso (base) va bene come associazione

OSS: Nella relazione ricorsiva le ENTITÀ sono 1 o 2? SONO 2 poiché dicono cose diverse

DIFF UML & E-R: nel modello E-R le relazioni sono insiemi e quindi non ci sono mai copie. Mentre in UML non è così per le associazioni

BISOGNA ENTATIZZARE LE UGUAGLIANZE E NON LE DIFFERENZE

OSS: Come lo introduco l'attributo TIPO? basta un attributo oppure devo utilizzare una classe?

Scelte: scrivere TIPO e descrivere esterno a questo sistema fo si che stiamo rimandando lo def ad un possibile futuro

Studio di caso: ERedit

(12 marzo 2010)

ERedit è un sistema per la gestione di diagrammi Entità-Relazione (ER). ERedit consente di rappresentare:

- Diagrammi ER, ciascuno con un nome che lo identifica e da un autore (che lo ha creato). Ad esempio, il diagramma *Università* creato da *Mario Rossi*.
- Un diagramma ER può contenere delle entità, ciascuna con un nome (che la identifica nell'ambito di un diagramma) ed una descrizione. Ad esempio, l'entità *Studente*, con descrizione *Studente di questa università*.
- Ad un'entità possono essere associati degli attributi, ciascuno con un nome (che lo identifica nell'ambito dell'entità), un tipo ed una descrizione. Ad esempio, l'attributo *Nome* di *Studente*, di tipo *String*, che è *Nome di questo studente*.
- Un diagramma ER può contenere delle relazioni binarie (ovvero, relazioni a cui partecipano due entità). Ciascuna relazione ha un nome (che la identifica nell'ambito di un diagramma) ed una descrizione. Ad esempio, la relazione *Iscrizione*, tra *Studente* e *Facoltà*, con descrizione *Lo studente è iscritto alla facoltà*. Sono ammesse anche relazioni binarie ricorsive, che coinvolgono due volte la stessa entità.
- Nota: non sono di interesse, al momento, né identificatori delle entità, né relazioni N-arie, né attributi delle relazioni, né cardinalità delle relazioni, né generalizzazioni-specializzazioni.

ERedit consente di creare e modificare diagrammi ER, mediante un certo numero di casi d'uso, dai quali sono state identificate le seguenti operazioni di sistema (oltre ad altre, che però non sono di interesse):

1. Creazione di un nuovo diagramma ER, dato il nome del diagramma e il nome dell'autore. Il diagramma è inizialmente vuoto. Da quel momento in poi, il diagramma ER creato è considerato il diagramma corrente.
2. Inserimento di una nuova entità nel diagramma corrente, dato il nome e la descrizione dell'entità.
3. Inserimento di una nuova relazione binaria nel diagramma corrente, dato il nome e la descrizione della relazione, nonché i nomi delle due entità coinvolte dalla relazione.
4. Inserimento di un nuovo attributo in una entità del diagramma corrente, dato nome, tipo e descrizione dell'attributo, nonché il nome dell'entità a cui l'attributo va associato.
5. Cancellazione di una relazione dal diagramma corrente, dato il suo nome. (Se il caso, insieme alla relazione vanno cancellati anche tutti i suoi attributi.)
6. Cancellazione di una entità dal diagramma corrente, dato il suo nome. Insieme all'entità vanno cancellati anche tutti i suoi attributi, nonché tutte le relazioni in cui l'entità è coinvolta.
7. Chiusura del diagramma corrente (che cessa di essere corrente).

Sono anche di interesse almeno le seguenti interrogazioni:

1. Visualizzazione dell'elenco di tutti i diagrammi.
2. Per un diagramma, visualizzazione di tutte le sue entità (ciascuna con i suoi attributi) e di tutte le sue relazioni (ciascuna con l'indicazione delle entità che vi partecipano).

Analisi e progettazione software

27/3/13
Lezione X

DAI REQUISITI ALLA PROGETTAZIONE

OOD: fa la cosa bene → progetta una soluzione che soddisfa i requisiti attualmente ai requisiti di quest'iterazione

→ i requisiti sono destinati a cambiare, per questo è addirittura opportuna, nelle iterazioni iniziali, che il cambiamento venga provocato

PRINCIPI DI PROGETTAZIONE:

- il progetto deve essere riconducibile al modello di analisi: SALTO rappresentazionale basso
- il progetto dei dati è importante come il progetto delle operazioni
- le interfacce deve essere progettato con cura ed è un obiettivo cruciale
- il progetto a livello di componenti deve essere funzionalmente coeso, indipendente
- i modelli di progetto devono essere comprensibili
- il progetto deve essere sviluppato in modo iterativo

ARCHITETTURA SOFTWARE

architettura software: un sistema SW è composto generalmente da numerosi classi.
Per gestire questa complessità è opportuno progettare anche un'organizzazione fondamentale per il sistema pochi elementi SW

"È l'insieme delle decisioni significative sull'organizzazione principale di un sistema software"

→ una tale organizzazione può essere descritta mediante i diagrammi dei PACKAGE: dove quest'ultimi rappresentano elementi a grana grossa

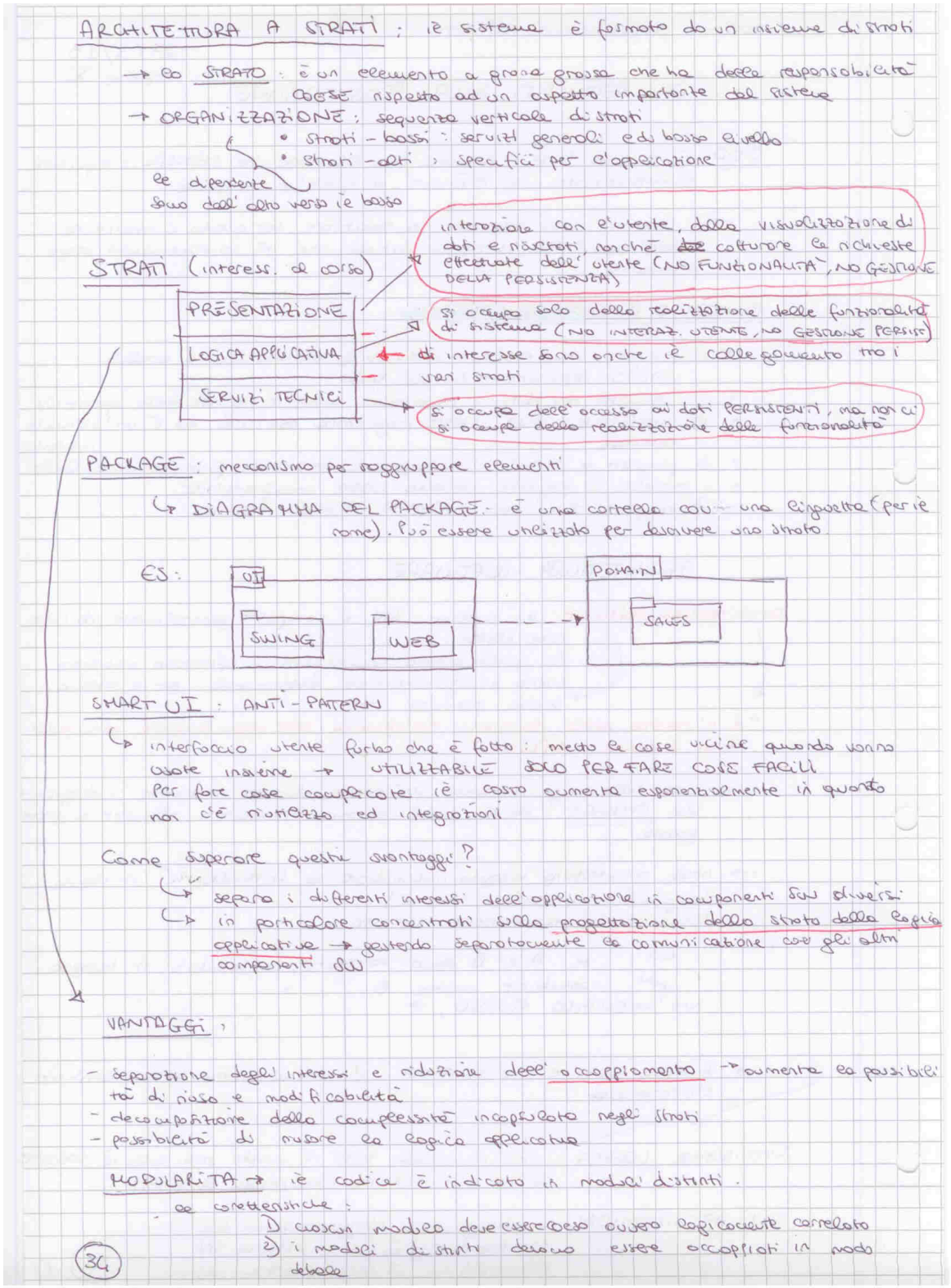
→ nella architettura bisogna individuare le DIPENDENZE → chiede l'erogazione di servizi (metodi)
Devono essere tenutamente separate:

ES: se A e B sono correlate, se cambio A bisogna non cambiare anche B
NO VOGLIAMO QUESTO

→ bisogna riuscire a cambiare l'implementazione senza cambiare l'interfaccia

Architettura LOGICA: non ci interessa tutto il codice ma solo il software con un'organizzazione concettuale a STRATI

STILI ARCHITETTURALI: che guidano l'organizzazione dell'architettura logica di un sistema SW prenderemo in considerazione un


ARCHITETTURA A STRATI : il sistema è formato da un insieme di strati
→ Lo STRATO : è un elemento a grana grossa che ha delle responsabilità coese rispetto ad un aspetto importante del sistema
→ ORGANIZZAZIONE : sequenza verticale di strati
• strati - bassi : servizi generali e di basso livello
• strati - alti : specifici per l'applicazione
le dipendenze sono dall'alto verso il basso
STRATI (interess. di corso)
PRESENTAZIONE
LOGICA APPLICATIVA
SERVIZI TECNICI
interazione con l'utente, della visualizzazione di dati e risultati, nonché delle cotture e richieste effettuate dell'utente (NO FUNZIONALITÀ, NO GESTIONE DELLA PERSISTENZA)
si occupa solo della realizzazione delle funzionalità di sistema (NO INTERAZ. UTENTE, NO GESTIONE PERSIS)
di interesse sono anche le collaborazioni tra i vari strati
si occupa delle accesso ai dati PERSISTENTI, ma non ci si occupa della realizzazione delle funzionalità
PACKAGE : meccanismo per raggruppare elementi
→ DIAGRAMMA DEL PACKAGE : è una cartella con una etichetta (per il nome). Può essere utilizzato per descrivere uno strato
ES :
UI
SWING
WEB
DOMAIN
SALES
SMART UI : ANTI - PATTERN
→ interfaccia utente furba che è fatto : mette le cose vicine quando vanno usate insieme → UTILIZZABILE SOLO PER FARE COSE FACILI
Per fare cose complicate il costo aumenta esponenzialmente in quanto non c'è riutilizzo ed integrazioni
Come superare questi svantaggi ?
→ separa i differenti interessi dell'applicazione in componenti SW diversi
→ in particolare concentrati sulla progettazione dello strato della logica applicativa → gestendo separatamente la comunicazione con gli altri componenti SW
VANTAGGI :
- separazione degli interessi e riduzione dell'accoppiamento → aumenta la possibilità di riuso e modificabilità
- decomposizione della complessità incapsulato negli strati
- possibilità di riuso della logica applicativa
MODULARITÀ → il codice è indicato in moduli distinti.
le caratteristiche :
1) ciascun modulo deve essere coeso ovvero logicamente correlato
2) i moduli di strati devono essere accoppiati in modo debole
34

Analisi e progettazione software

27/3/13
Lezione X

STRATO DELLA LOGICA APPLICATIVA:

↳ come può essere progettato?

- tutto in una classe
- decomposizione COMPORTAMENTALE / FUNZIONALE
  partizionamento delle responsabilità
  - una classe per ciascuna operazione - CASO D'USO
  - stile di controllo accentrato
- Decomposizione RAPPRESENTAZIONALE: guidata dalle informazioni
  partizionamento delle responsabilità
  - classi sui ispirati agli oggetti del dominio (ecco perché anche di (DOMINIO))
  - stile di controllo distribuito
  - salto rappresentazionale basso

↳ diverse strategie nell'organizzazione logica applicativa
1) APPROCCIO TRASAZIONALE
2) PROCEDURALE
3) Blocks in MODEL:

→ LIVELLO: indica un nodo fisico
→ STRATO: una fetta (verticale) dell'architettura
→ PARTIZIONE: porzione di uno strato

PRINCIPIO DI SEPARAZIONE MODELLO-VISTA
gli oggetti e le classi del modello (dei due strati più bassi) non devono essere accoppiati direttamente alla vista

INTERPRETAZIONI DELLE OPERAZIONI DI SISTEMA
ogni operazione del piano di sopra invocherà un metodo del piano di sotto

PROGETTAZIONE A OGGETTI

3/4/13
Lezione XI

→ Quando viene fatta la progettazione a oggetti?
↳ fare la progettazione mentre si sta scrivendo il codice
↳ disegno (UML) e poi codifica, progr.
↳ solo disegno

nel nostro caso è preferibile scrivere tanto codice e fare pochi disegni

più sono difficili → utilizzo in maniera ~~più pesante~~ pesante e intensa di disegni / diagrammi
più sono facili → utilizzo concorrentemente codice + parte visiva

→ Quanto tempo devo dedicare?
un'iterazione dura 3 settimane
- dedico poche ore al disegno peggiore
- passo poi all'implementazione e test
  - prog. durante la codifica

abbiamo abbandonato il pto di vista concettuale, da adesso sono un oggetto SOFTWARE

Nella progettazione si utilizzano:

1) MODELLI STATICI: descrivono l'operazione, indica un pezzo di codice → ci danno l'idea di organizzazione del codice
2) MODELLI DINAMICI: indica l'occupazione di certe aree di memoria

i diagrammi più importanti a prendere decisioni di progetto sono i diagrammi per i modelli dinamici

oss: bisogna utilizzarci entrambi in maniera parallela

progettazione richiede:
- PRINCIPI ASSEGNAZIONE RESPONSABILITÀ
- DESIGN PATTERN

UML: PENSARE ad oggetti

una capacità critica nello sviluppo OO è saper assegnare in modo oculato responsabilità agli oggetti software

DIAGRAMMI DI INTERAZIONE (NOTAZIONE): descrivono oggetti che interagiscono tra loro scambiandosi messaggi, e richieste di operazioni/funzionalità

- un'interazione è la specifica di come scambiare messaggi tra oggetti per eseguire un compito nell'ambito di un certo contesto

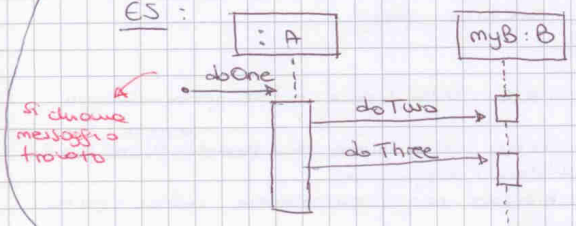
2 tipi di diagrammi di INTERAZIONE

1) DIAGRAMMI DI SEQUENZA: mostra l'interazione tra un insieme di oggetti a partire da un messaggio TROVATO

- diagramma bidirezionale, orientato, tradizionalmente è presente l'asse del tempo (y) mentre l'asse orizzontale ci sono le operazioni/messaggi scambiati

oss: in UML le parentesi non sono obbligatorie, se non passo parametri

ES:

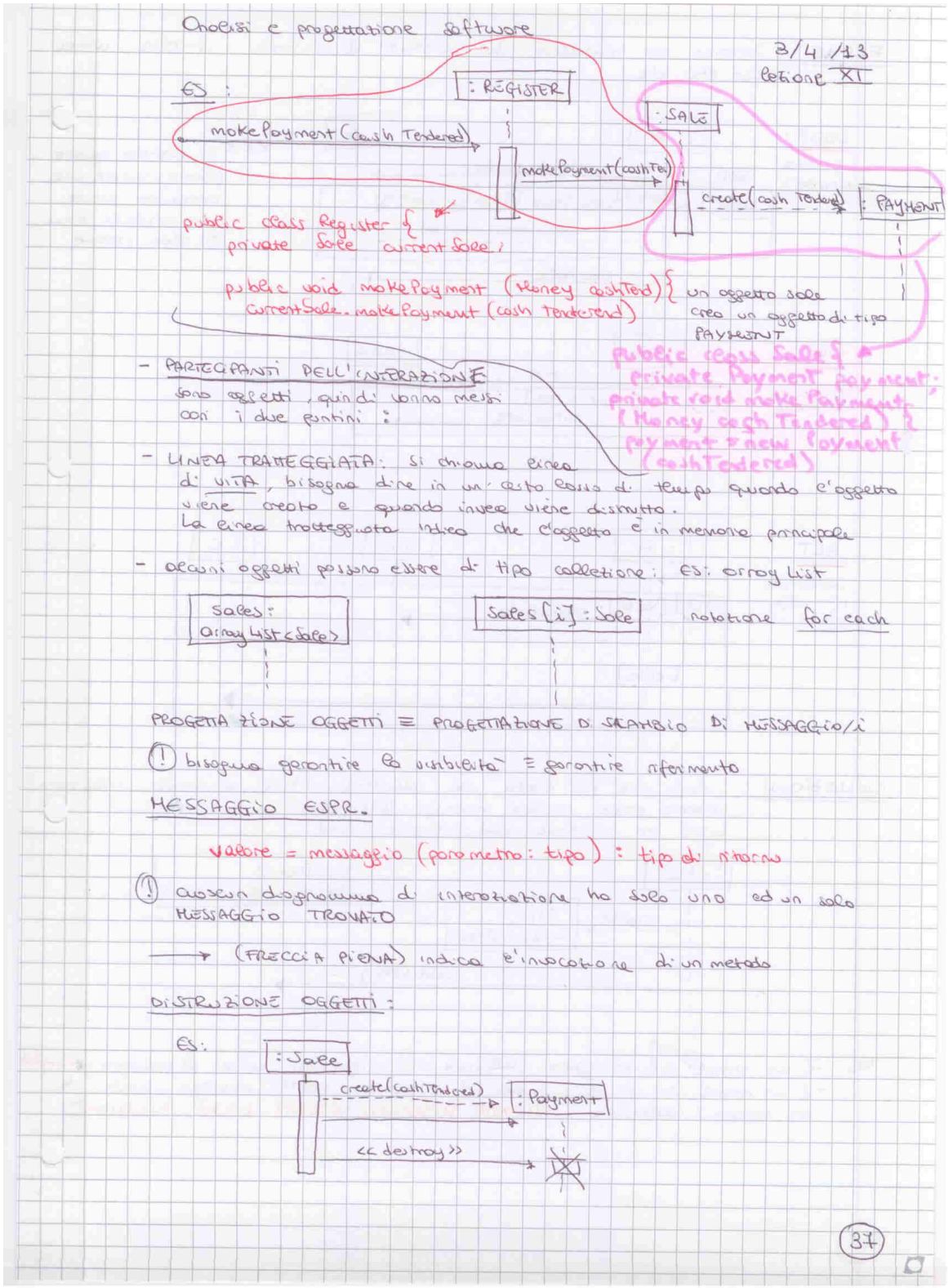


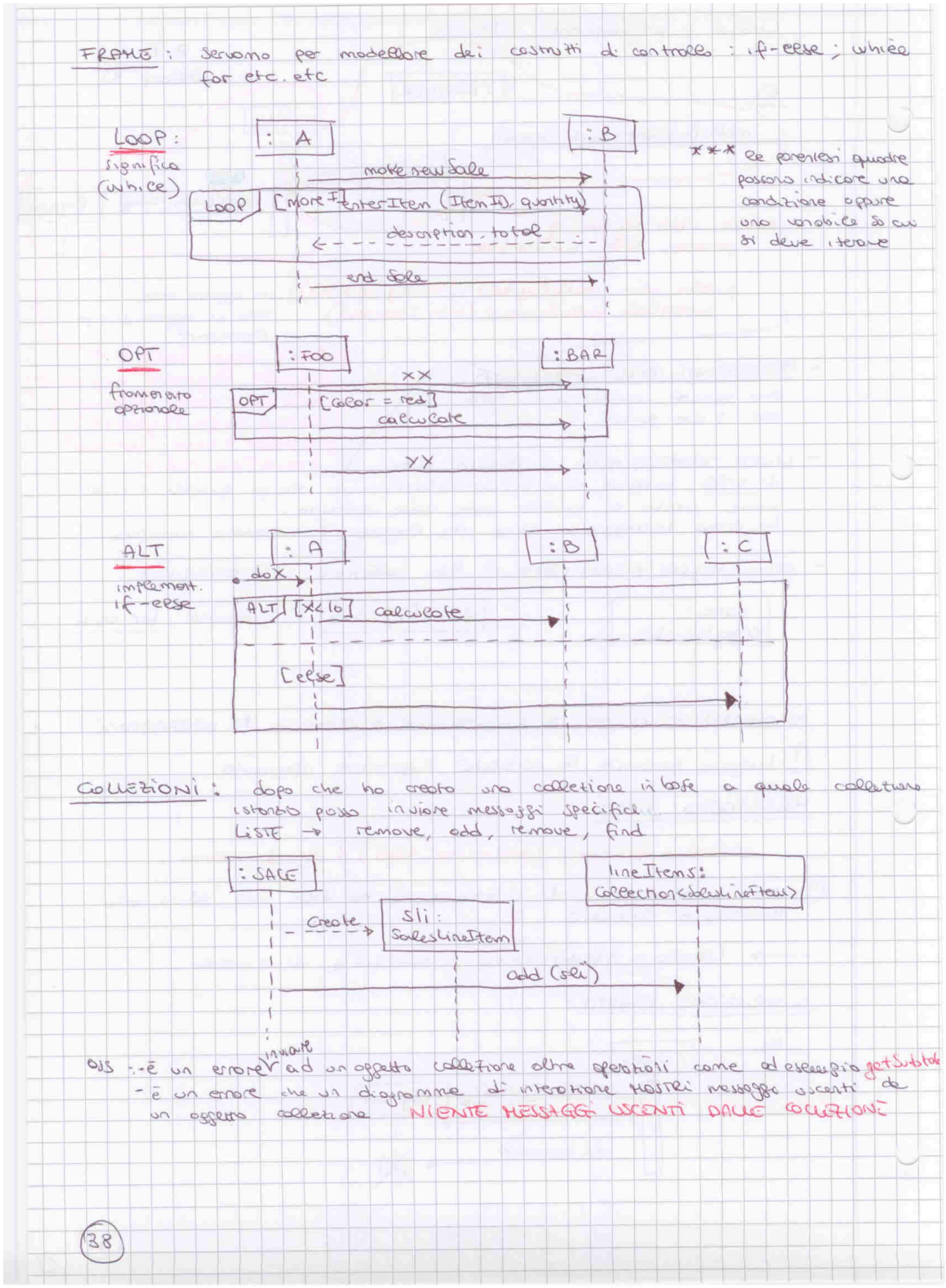
- nel codice avremo una classe che si chiama A ed una che si chiama B
- è lecito fare doOne solo se nella classe A è presente il suddetto metodo.
- un oggetto deve conoscere un riferimento per invocare i metodi dell'altro oggetto

FORZE: mostra chiaramente la sequenza temporale dello scambio dei messaggi
- opzioni di notazione numerose e dettagliate

DEBOLEZZE: oggetti devono essere disposti orizzontalmente

CONTROLLER: oggetto responsabile di ricevere le operazioni di sistema


Analisi e progettazione software
3/4/13
lezione XI
ES:
: REGISTER
: SALE
: PAYMENT
makePayment(cashTendered)
makePayment(cashTen)
create(cash Tendered)
public class Register {
private Sale currentSale;
public void makePayment (Money cashTend) {
currentSale.makePayment (cashTendered)
un oggetto Sale crea un oggetto di tipo PAYMENT
public class Sale {
private Payment payment;
private void makePayment (Money cashTendered) {
payment = new Payment (cashTendered)
- PARTECIPANTI DELL'INTERAZIONE
sono oggetti, quindi vanno messi con i due puntini :
- LINEA TRATTEGGIATA: si chiama linea di VITA, bisogna dire in un certo senso di tempo quando c'oggetto viene creato e quando invece viene distrutto.
La linea tratteggiata indica che c'oggetto è in memoria principale
- alcuni oggetti possono essere di tipo collezione: ES: array list
Sales: ArrayList<Sale>
Sales[i]: Sale
notazione for each
PROGETTAZIONE OGGETTI ≡ PROGETTAZIONE DI SCAMBIO DI MESSAGGIO/I
! bisogna garantire la visibilità ≡ garantire riferimento
MESSAGGIO ESPR.
valore = messaggio (parametro: tipo) : tipo di ritorno
! ciascun diagramma di interazione ha solo uno ed un solo MESSAGGIO TROVATO
→ (FRECCIA PIENA) indica l'invocazione di un metodo
DISTRUZIONE OGGETTI:
ES:
: Sale
create(cashTendered)
: Payment
<< destroy >>

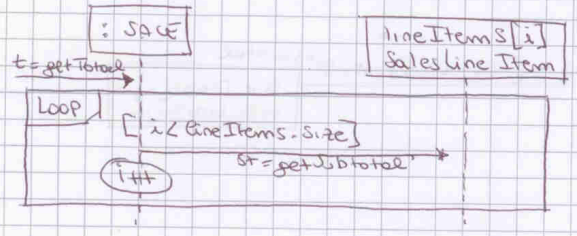

FRAME : servono per modellare dei costrutti di controllo : if-else ; while for etc. etc
LOOP : significa (while)
: A
: B
make newSale
LOOP [moreItems] enterItem (ItemID, quantity)
description, total
end Sale
*** se potessi quadre posso indicare una condizione oppure una variabile su cui si deve iterare
OPT
frammento opzionale
: FOO
: BAR
xx
OPT [color = red]
calculate
yy
ALT
implement. if-else
: A
: B
: C
doX
ALT [x<10] calculate
[else]
COLLEZIONI : dopo che ho creato una collezione in base a quale collezione istanzio posso inviare messaggi specifici
LISTE → remove, add, remove, find
: SALE
lineItems: Collection<SalesLineItem>
create
sli: SalesLineItem
add (sli)
OSS - è un errore inviare ad un oggetto collezione altre operazioni come ad esempio getSubtotal
- è un errore che in diagramma di interazione mostrei messaggi uscenti da un oggetto collezione NIENTE MESSAGGI USCENTI DALLE COLLEZIONI

Analisi e progettazione software

3/4/13
Lezione XI

COLLEZIONI

↳ ITERAZIONI SULLE COLLEZIONE




```
public class Sale {
  private List<SalesLineItem>
    lineItems;
  public Money getTotal() {
    Money total = new Money();
    Money subtotal;
    for (SalesLineItem lineItem:
        lineItems)
      subtotal = lineItem.getSub()
      total                      } return total;}
```

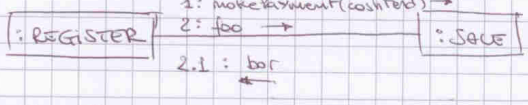
OSS: in questo corso si itererà sempre su tutti gli elementi di una collezione

OSS: in UML tutto è opzionale → qui non ci sono i box di attivazione perché si vuole mettere l'attenzione su altro

2) DIAGRAMMI DI COMUNICAZIONE:

COLLEGAMENTO: è un percorso di connessione tra 2 oggetti; quando disegnamo questo linee bisogna chiederci se c'è una referenziazione

ES:

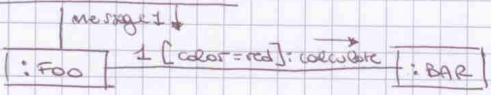


la numerazione esprime la relazione causa - effetto

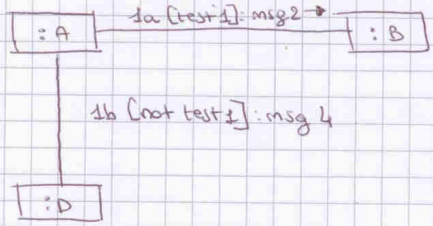
① collegamento e freccia sono elementi grafici distinti

OSS: il MESSAGGIO TROVATO non è numerato

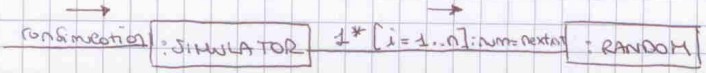
MESSAGGIO CONDIZIONALE:

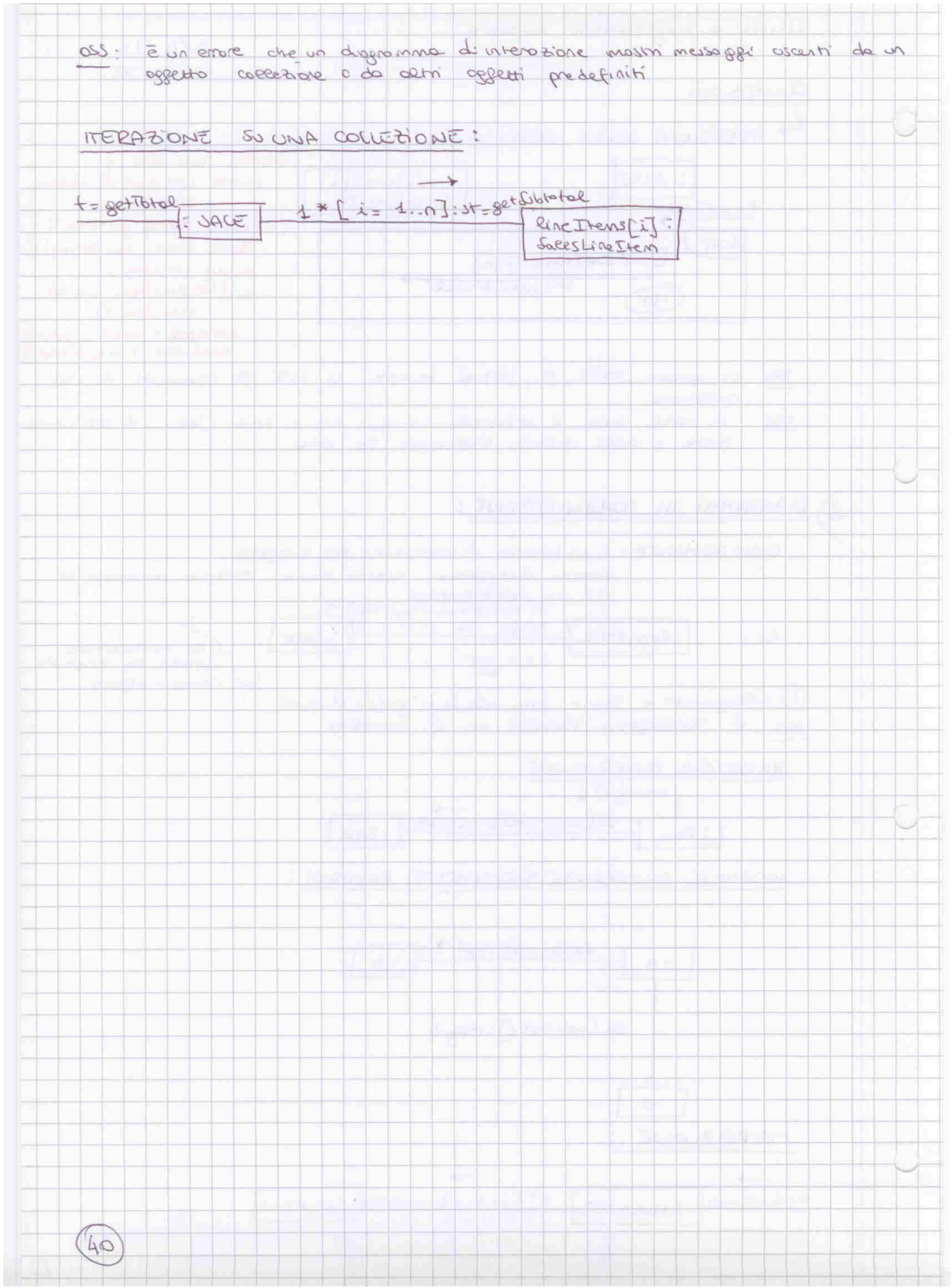


MESSAGGI CONDIZIONALI MUTUALMENTE ESCLUSIVI:



ITERAZIONE:




oss: è un errore che un diagramma di interazione mostri messaggi uscenti da un oggetto collezione o da altri oggetti predefiniti
ITERAZIONE SU UNA COLLEZIONE:
t = getTotal
: JAVE
1 * [i = 1..n]: st = getSubtotal
lineItems[i] : SalesLineItem
40

Analisi e progettazione del software
Anno Accademico 2012-2013
Scuola delle Arti – Requisiti

La ***Scuola delle Arti Acme*** (nel seguito chiama semplicemente Scuola) è una scuola che offre diplomi annuali in diverse discipline artistiche (ad es., in *Fotografia*). La Scuola usa il sistema software **AcmeArti** per la gestione delle informazioni relative ai corsi offerti, ai docenti e agli studenti, nonché ai piani formativi individuali (una specie di piani di studio) scelti dai singoli studenti.
Il sistema è usato principalmente dai seguenti tipi di attori: *docenti*, per dare la propria disponibilità a tenere corsi; *studenti*, per consultare i corsi offerti dalla scuola e per compilare i propri piani formativi individuali; e la *segretaria*, per consultare i piani formativi individuali scelti dagli studenti.

Alcune regole di dominio:

- La Scuola offre diversi diplomi, di durata annuale (ad es., *Fotografia*, *Grafica* e *Recitazione*).
- Nell'ambito di ciascun diploma (ad es., *Fotografia*) la Scuola offre ai suoi studenti un certo numero di corsi di durata mensile. Ad es., un corso di *Ritratto* (da 10 crediti) e un corso di *Paesaggio* (da 5 crediti).
- Ai fini del conseguimento del diploma, ciascuno studente deve frequentare corsi per 50 crediti.
- Prima dell'inizio di ciascun anno scolastico (nel corso del mese di settembre), ciascun docente della Scuola usa il sistema (caso d'uso UC1) per dare la propria disponibilità ad tenere una o più edizioni dei corsi tra quelli di sua competenza. Ad esempio, il docente *Rossi* potrebbe dare la sua disponibilità a tenere il corso di *Ritratto* nel mese di *febbraio* e il corso di *Reportage* nel mese di *aprile*.
- Per uno stesso corso possono essere previste più edizioni, tenute in mesi differenti, anche da docenti diversi. Ad esempio, il docente *Verdi* potrebbe dare la sua disponibilità a tenere il corso di *Ritratto* nel mese di *marzo*.
- All'inizio di ciascun anno scolastico (all'inizio del mese di ottobre), ciascuno studente della Scuola usa il sistema (caso d'uso UC2) per compilare il suo piano formativo individuale (abbreviato in PFI). Un PFI è l'elenco dei corsi che lo studente vuole frequentare per conseguire il diploma. Uno studente può scegliere nel suo PFI solo corsi offerti nell'ambito del diploma a cui è iscritto e per i quali è prevista nell'anno almeno un'edizione. Ad esempio, lo studente *Lombardo* potrebbe scegliere nel proprio PFI il corso di *Ritratto*, edizione di *febbraio*, tenuta dal docente *Rossi*. Uno studente non può scegliere nel proprio PFI più edizioni di uno stesso corso. Anche se per conseguire il titolo uno studente deve frequentare corsi per 50 crediti, il suo PFI deve contenere la scelta di corsi per 60 crediti (anche con l'indicazione di priorità da parte dello studente).
- Al termine del periodo di compilazione dei piani formativi individuali, la Scuola userà le scelte fatte dagli studenti per stabilire quali edizioni dei corsi dovranno essere effettivamente tenute nell'anno scolastico, e in quali mesi. Verranno adottati criteri di efficacia ed efficienza (ad esempio, ciascuna edizione di un corso dovrà normalmente avere tra 5 e 15 studenti). Inoltre sarà la Scuola a decidere quali corsi dovrà frequentare ciascuno studente. Verranno prese in considerazione le priorità espresse dagli studenti. *Nota: L'iterazione corrente non è interessata direttamente a quanto descritto in quest'ultimo punto. E' però interessata alla raccolta dei PFI degli studenti, nonché a consentire alla segreteria l'accesso a questi PFI. Quest'ultimo punto è stato scritto soprattutto per descrivere l'uso che si intende fare dei PFI.*

Analisi e progettazione del software
Anno Accademico 2012-2013
Scuola delle Arti – Requisiti

L'uso del sistema in discussione è descritto principalmente dai seguenti casi d'uso UC1-UC4 (di cui è di interesse sia lo scenario principale di successo che ogni estensione mostrata):

Caso d'uso UC1: Inserimento edizioni corsi – Attore primario: un Docente.
Scenario principale di successo:
1. Il Docente vuole dare la disponibilità per tenere le edizioni di uno o più corsi.
2. Il Docente inserisce il suo codice identificativo e la sua password. Il Sistema verifica la correttezza dei dati immessi, e autentica il Docente. Il Sistema mostra nome e cognome del Docente.
3. Il Docente sceglie l'attività "Inserimento edizioni corsi". Il Sistema mostra l'elenco dei corsi di competenza per il Docente, mostrando per ciascun corso il codice e il nome.
4. Il Docente inserisce il codice di un corso per dare la propria disponibilità a tenere un'edizione di quel corso. Il Sistema mostra le informazioni dettagliate relative al corso selezionato (nome, descrizione e numero di crediti).
5. Il Docente inserisce il mese in cui intende tenere un'edizione del corso selezionato, insieme a una descrizione del programma che intende utilizzare in quell'edizione del corso. Il Sistema registra la nuova edizione del corso, con la disponibilità del Docente a tenerla.

Il Docente ripete i passi 4-5 fino a che non indica che ha terminato.

6. Il Sistema visualizza un riepilogo delle edizioni dei corsi che il Docente è disponibile a tenere, ciascuno con il relativo programma e mese.

Caso d'uso UC2: Compilazione Piano formativo individuale – Attore primario: uno Studente.
Scenario principale di successo:
1. Lo Studente vuole compilare il proprio piano formativo individuale (PFI).
2. Lo Studente inserisce il suo codice identificativo e la sua password. Il Sistema verifica la correttezza dei dati immessi, e autentica lo Studente. Il Sistema mostra nome e cognome dello Studente, e il nome del diploma a cui è iscritto.
3. Lo Studente sceglie l'attività "Compilazione piano formativo individuale".
4. Il Sistema mostra l'elenco dei corsi offerti nell'ambito del diploma a cui lo Studente è iscritto, mostrando per ciascun corso il codice e il nome.
5. Lo Studente inserisce il codice di un corso che vuole inserire nel proprio PFI. Il Sistema mostra ulteriori informazioni sul corso scelto (nome, descrizione e numero di crediti) nonché l'elenco delle edizioni previste per quel corso (con indicazione del docente, del mese e del programma del corso).
6. Lo Studente inserisce il codice di una delle edizioni del corso scelto, per inserire il corso nel proprio PFI, nonché una priorità associata a questa scelta (*alta*, *media* o *bassa*). Il Sistema registra la scelta dello Studente.

Lo Studente ripete i passi 5-6 fino a che non indica che ha terminato.

7. Il Sistema mostra un riepilogo del PFI inserito dallo Studente (con un elenco dei corsi scelti, ciascuno con l'indicazione dell'edizione scelta, del docente e del mese, e della priorità specificata, nonché un totale dei crediti scelti nel PFI).
8. Lo Studente conferma l'inserimento del proprio PFI. Il Sistema registra il nuovo PFI, registrando anche la data e l'ora di inserimento, e genera una ricevuta.
9. Lo Studente prende la ricevuta e va via.

Estensioni:
4-7a Lo Studente annulla l'operazione di inserimento. Il Sistema non registra nessun nuovo PFI.

Caso d'uso UC3: Consultazione PFI di uno studente – Attore primario: la Segreteria.
La Segreteria usa il Sistema per visualizzare il PFI di un certo studente. Il Sistema mostra i dati dello studente e, per ciascun corso nel suo PFI, il nome del corso, l'edizione scelta (docente e mese) e la priorità specificata, nonché il totale dei crediti scelti.

Caso d'uso UC4: Consultazione scelte degli studenti – Attore primario: la Segreteria.
La Segreteria usa il Sistema per visualizzare, per una certa edizione di un corso, quali sono gli studenti che hanno scelto quell'edizione del corso nel proprio PFI.

Analisi e progettazione del software
Anno Accademico 2012-2013
Homework 1 – Analisi – Modellazione di dominio

Regole: **SCRIVERE IN MODO LEGGIBILE E NON AMBIGUO**. Scrivere il proprio nome su ciascun foglio utilizzato, in alto a destra. Accanto allo svolgimento di ciascun esercizio o domanda va indicato chiaramente l'esercizio o la domanda a cui lo svolgimento è relativo (ad esempio, Esercizio A1, diagramma di oggetti di dominio, oppure Esercizio A2, domanda A2.2). Alle domande in cui è prevista una risposta binaria SI/NO ("è opportuno fare questo?") è necessario scrivere in modo esplicito prima la risposta alla domanda (SI oppure NO) e poi dare una breve motivazione della risposta.

In questo homework si faccia riferimento ai requisiti per il sistema **AcmeArti**, descritti in un documento separato.

Esercizio A1 (Modellazione di dominio)
Fare l'analisi a oggetti per il sistema in discussione, come segue:

- Mostrare il modello di dominio, relativo a tutti casi d'uso mostrati (UC1-UC4).
- Mostrare inoltre un diagramma di oggetti di dominio che rappresenta:
  - un corso di *Ritratto*, per il diploma in *Fotografia*, per il quale sono previsti un'edizione a *febbraio* (tenuta dal docente *Rossi*) e una a *marzo* (tenuta dal docente *Verdi*);
  - un corso di *Paesaggio*, per il diploma in *Fotografia*, per il quale è prevista un'edizione a *dicembre* (tenuta dal docente *Verdi*);
  - uno studente *Lombardo*, del diploma in *Fotografia*, che nel proprio piano formativo individuale ha scelto i corsi di *Ritratto* (edizione di *febbraio*, priorità *alta*) e *Paesaggio* (edizione di *dicembre*, priorità *media*);
  - uno studente *Toscano*, del diploma in *Fotografia*, che nel proprio piano formativo individuale ha scelto i corsi di *Ritratto* (edizione di *marzo*) e *Paesaggio* (edizione di *dicembre*), entrambi con priorità *alta*.

Esercizio A2 (Domande)
Relativamente al precedente esercizio A1, rispondere alle seguenti domande (rispondere in modo sintetico, scrivendo circa 3-5 righe per ciascuna domanda):

- (A2.1) E' opportuno introdurre nel modello di dominio una classe concettuale chiamata "Corso"? Motivare la risposta, discutendo anche l'eventuale significato attribuito alla classe.
- (A2.2) E' opportuno introdurre nel modello di dominio una classe concettuale chiamata "Edizione Corso"? Motivare la risposta, discutendo anche l'eventuale significato attribuito alla classe.
- (A2.3) E' opportuno introdurre nel modello di dominio una classe concettuale chiamata "Diplomi"? Motivare la risposta, discutendo anche l'eventuale significato attribuito alla classe.
- (A2.4) Per rappresentare i piani formativi individuali degli studenti, è sufficiente utilizzare un'associazione tra "Studente" e "Corso", o qualcosa del genere? Oppure è opportuno introdurre nel modello di dominio una classe concettuale chiamata ad esempio "Piano formativo individuale"? O addirittura è necessaria anche una classe concettuale chiamata ad esempio "Riga di Piano formativo individuale"? Motivare la risposta, discutendo anche l'eventuale significato attribuito alle classi introdotte con lo scopo specifico di rappresentare i piani formativi individuali degli studenti.

Analisi e progettazione del software
Anno Accademico 2012-2013
Scuola delle Arti – Requisiti

L'uso del sistema in discussione è descritto principalmente dai seguenti casi d'uso UC1-UC4 (di cui è di interesse sia lo scenario principale di successo che ogni estensione mostrata):

Caso d'uso UC1: Inserimento edizioni corsi – Attore primario: un Docente.
Scenario principale di successo:
1. Il Docente vuole dare la disponibilità per tenere le edizioni di uno o più corsi.
2. Il Docente inserisce il suo codice identificativo e la sua password. Il Sistema verifica la correttezza dei dati immessi, e autentica il Docente. Il Sistema mostra nome e cognome del Docente.
3. Il Docente sceglie l'attività "Inserimento edizioni corsi". Il Sistema mostra l'elenco dei corsi di competenza per il Docente, mostrando per ciascun corso il codice e il nome.
4. Il Docente inserisce il codice di un corso per dare la propria disponibilità a tenere un'edizione di quel corso. Il Sistema mostra le informazioni dettagliate relative al corso selezionato (nome, descrizione e numero di crediti).
5. Il Docente inserisce il mese in cui intende tenere un'edizione del corso selezionato, insieme a una descrizione del programma che intende utilizzare in quell'edizione del corso. Il Sistema registra la nuova edizione del corso, con la disponibilità del Docente a tenerla.

Il Docente ripete i passi 4-5 fino a che non indica che ha terminato.

6. Il Sistema visualizza un riepilogo delle edizioni dei corsi che il Docente è disponibile a tenere, ciascuno con il relativo programma e mese.

Caso d'uso UC2: Compilazione Piano formativo individuale – Attore primario: uno Studente.
Scenario principale di successo:
1. Lo Studente vuole compilare il proprio piano formativo individuale (PFI).
2. Lo Studente inserisce il suo codice identificativo e la sua password. Il Sistema verifica la correttezza dei dati immessi, e autentica lo Studente. Il Sistema mostra nome e cognome dello Studente, e il nome del diploma a cui è iscritto.
3. Lo Studente sceglie l'attività "Compilazione piano formativo individuale".
4. Il Sistema mostra l'elenco dei corsi offerti nell'ambito del diploma a cui lo Studente è iscritto, mostrando per ciascun corso il codice e il nome.
5. Lo Studente inserisce il codice di un corso che vuole inserire nel proprio PFI. Il Sistema mostra ulteriori informazioni sul corso scelto (nome, descrizione e numero di crediti) nonché l'elenco delle edizioni previste per quel corso (con indicazione del docente, del mese e del programma del corso).
6. Lo Studente inserisce il codice di una delle edizioni del corso scelto, per inserire il corso nel proprio PFI, nonché una priorità associata a questa scelta (*alta*, *media* o *bassa*). Il Sistema registra la scelta dello Studente.

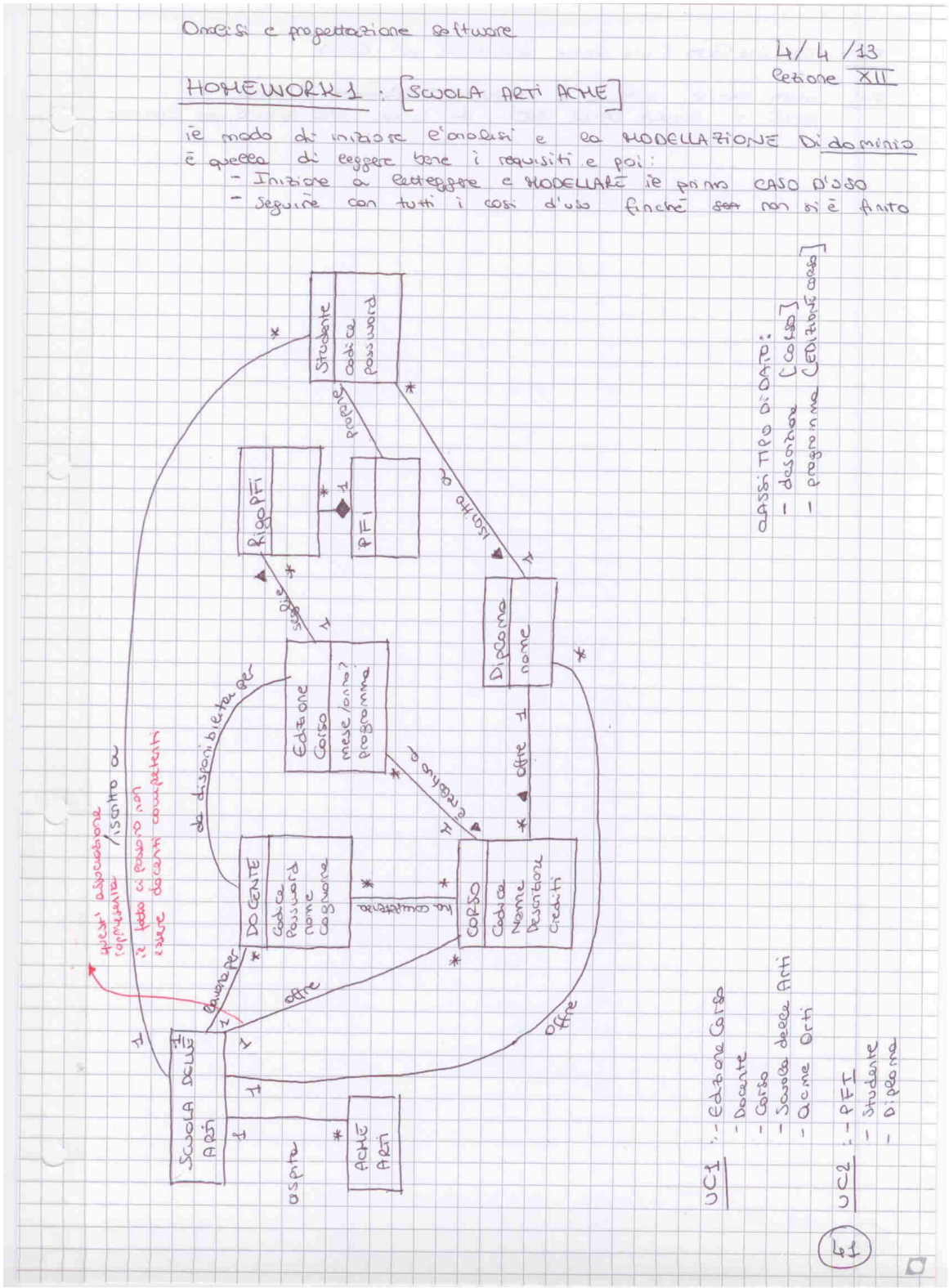
Lo Studente ripete i passi 5-6 fino a che non indica che ha terminato.

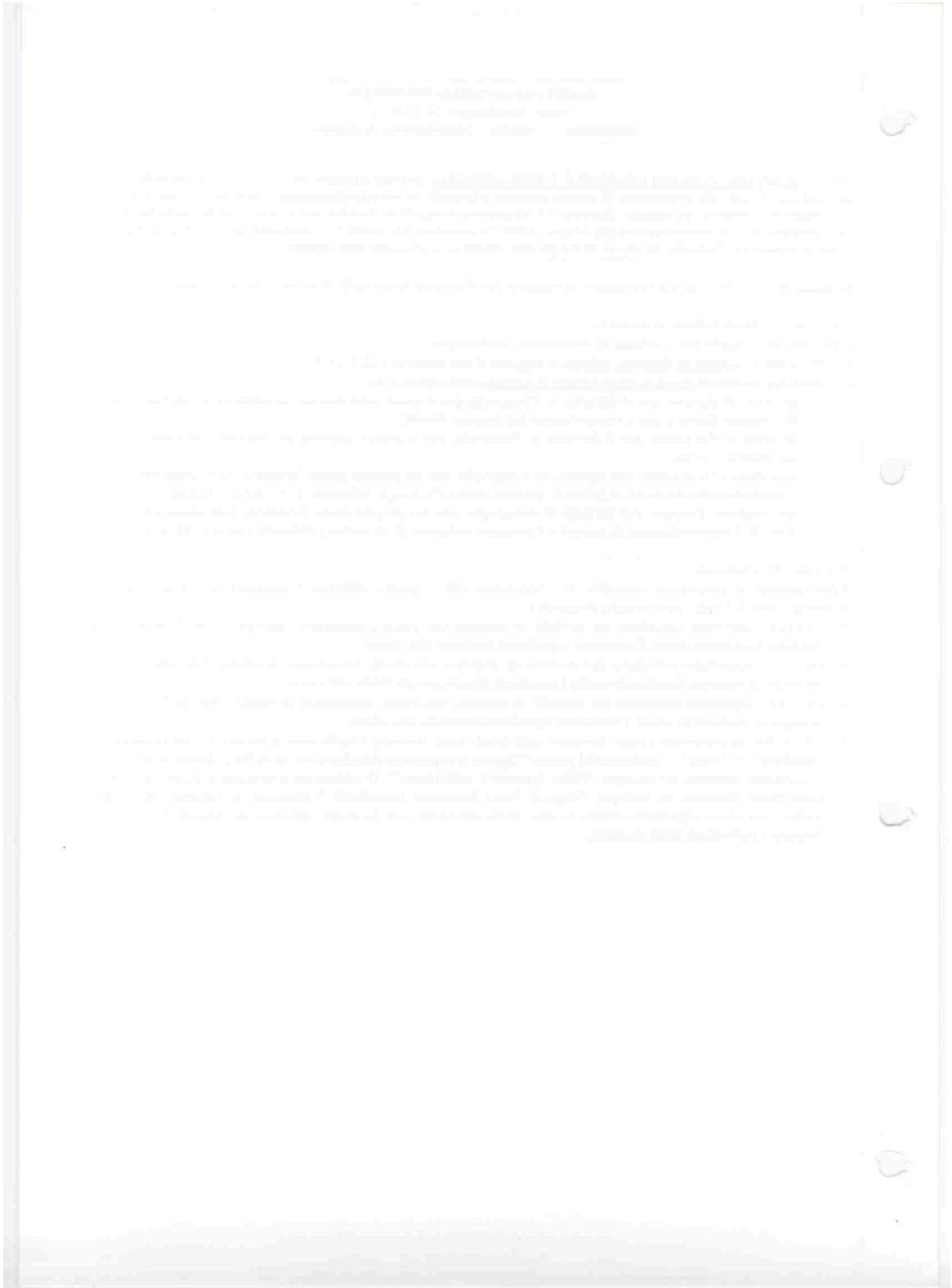
7. Il Sistema mostra un riepilogo del PFI inserito dallo Studente (con un elenco dei corsi scelti, ciascuno con l'indicazione dell'edizione scelta, del docente e del mese, e della priorità specificata, nonché un totale dei crediti scelti nel PFI).
8. Lo Studente conferma l'inserimento del proprio PFI. Il Sistema registra il nuovo PFI, registrando anche la data e l'ora di inserimento, e genera una ricevuta.
9. Lo Studente prende la ricevuta e va via.

Estensioni:
4-7a Lo Studente annulla l'operazione di inserimento. Il Sistema non registra nessun nuovo PFI.

Caso d'uso UC3: Consultazione PFI di uno studente – Attore primario: la Segreteria.
La Segreteria usa il Sistema per visualizzare il PFI di un certo studente. Il Sistema mostra i dati dello studente e, per ciascun corso nel suo PFI, il nome del corso, l'edizione scelta (docente e mese) e la priorità specificata, nonché il totale dei crediti scelti.

Caso d'uso UC4: Consultazione scelte degli studenti – Attore primario: la Segreteria.
La Segreteria usa il Sistema per visualizzare, per una certa edizione di un corso, quali sono gli studenti che hanno scelto quell'edizione del corso nel proprio PFI.


Analisi e progettazione software
4/4/13
Lezione XII
HOMEWORK 1 - [SCUOLA ARTI ACME]
Studente
codice
password
Progetto
RigaPFI
PFI
Diploma
nome
Edizione
Corso
mese/anno
programma
DOCENTE
codice
password
nome
cognome
CORSO
codice
Nome
descrizione
crediti
SCUOLA DELLE ARTI
ACME ARTI
CLASSI TIPO DI DATI:
- docente [corso]
- programma [edizione corso]
UC1 :- Edizione Corso
- Docente
- Corso
- Scuola delle Arti
- Acme Arti
UC2 :- PFT
- Studente
- Diploma



Analisi e progettazione software

8/4/13
Lezione XIII

DIAGRAMMI DELLE CLASSI DI UML

mostrano classi, interfacce, associazioni → viene preso in considerazione soprattutto il punto di vista software

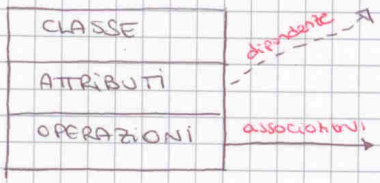
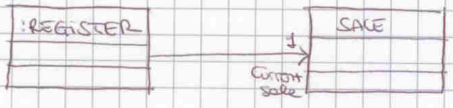


DIAGRAMMA DELLE CLASSI DI PROGETTO (DCD):

sono diagrammi delle classi viste dal p.to di vista software

- nei (DCD) le associazioni "frecce" sono orientate e si mette eo mette pericita sopra alla freccia di navigabilità

ES:



questo significa che nella classe register è stata fatta di una variabile d'istanza sale

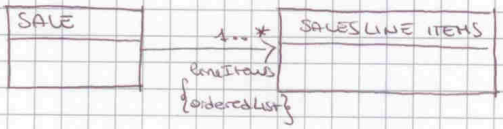
CLASSIFICATORE (UML): elemento di modello che descrive caratteristiche

- COMPORTAMENTALI: (operazioni)
- STRUTTURALI: sono gli attributi e le associazioni, in pratica le proprietà strutturali di una classe sono quelle che andranno a rappresentare da VARIABILI D'ISTANZA

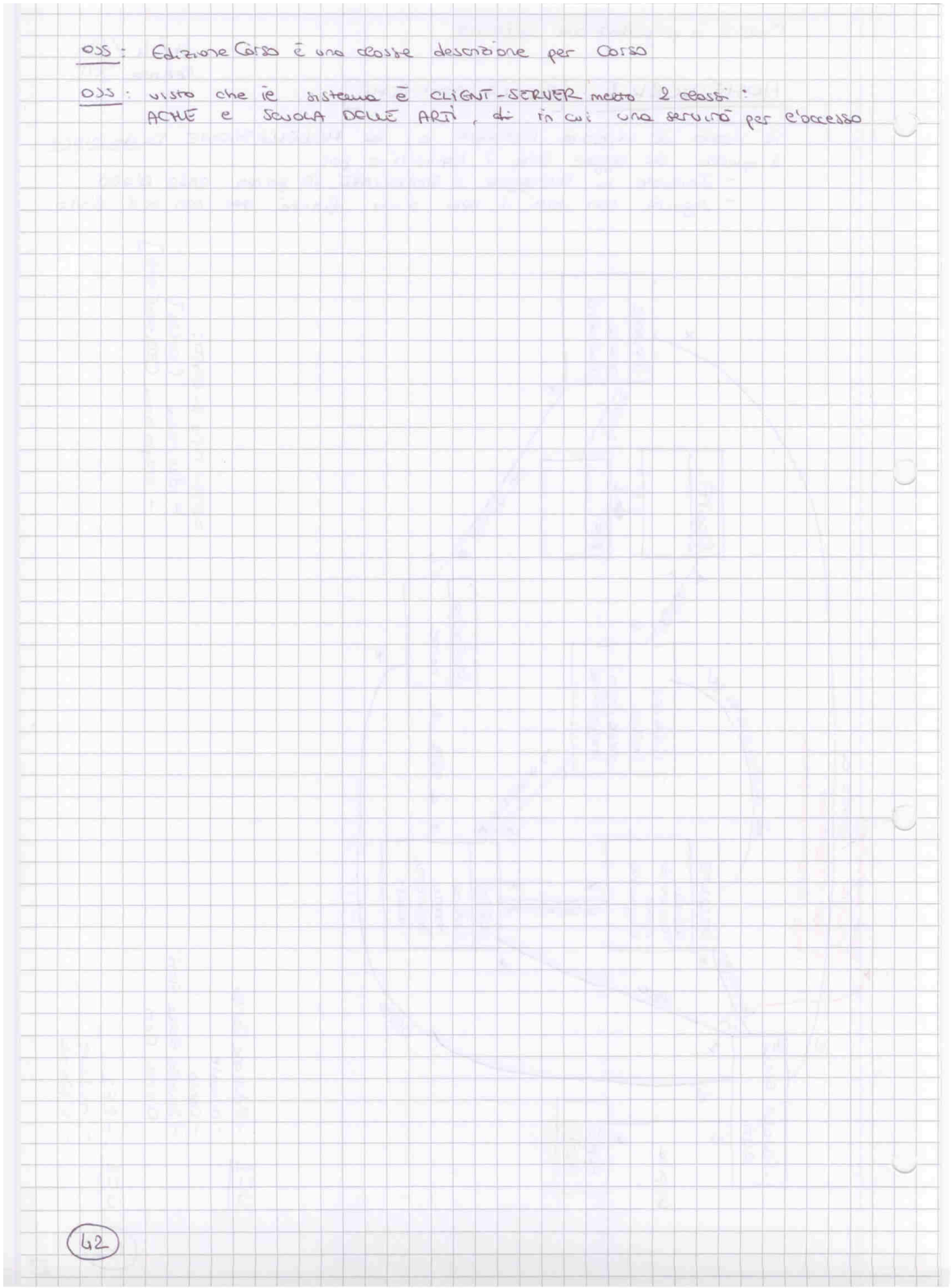
oss: se nella associazione non devo mettere l'attributo nella classe REGISTER

UTILIZZO ATTRIBUTO: <=> 1) il tipo è un tipo di dato
2) un tipo di dato è un insieme di valori in cui l'identità univoca non è significativa

ES:



! progettare significa capire quali oggetti implementare e quali messaggi si scambiano.


OSS: Edizione Corso è una classe descrizione per Corso
OSS: visto che il sistema è CLIENT-SERVER mette 2 classi:
ACME e SCUOLA DELLE ARTI, di in cui una servirà per l'accesso
42

Analisi e progettazione software

8/4/13
Lezione XIII

GRASP: PROGETTAZIONE DI OGGETTI CON RESPONSABILITÀ

PROGETTAZIONE GUIDATA DALLE RESPONSABILITÀ: bisogna pensare in termini di responsabilità (RDD)

Responsabilità: astrazione di ciò che rappresenta una componente software

OSS: se non assegno responsabilità non scrivo una riga di codice

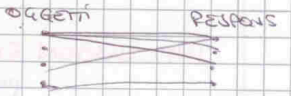
- di FARE: fare qualcosa esso stesso, richiedere ad altri oggetti di eseguire azioni → coordinare le attività di altri oggetti (dei contratti dei diagram. di seq.)
- di CONOSCERE: - conoscere i propri dati privati incapsulati
  - conoscere gli oggetti correlati
  - conoscere cose che può derivare o calcolare (dal modello di domin)

le responsabilità sono a grado piccolo medio nel nostro caso

Collaborazione: un oggetto a cui è stata assegnata una responsabilità, ed deve poi soddisfare
- un oggetto per soddisfare una responsabilità può agire da solo oppure può collaborare con altri oggetti

2 approcci per la progettazione guidata dalle responsabilità (RDD)

1) - In primis identifica gli oggetti
   - identifica le responsabilità
   - assegna le responsabilità agli oggetti



2) - identifica la responsabilità (considerala una alla volta)
   - chiediti quale oggetto assegnare questa responsabilità
     - tra gli oggetti già identificati, oppure identificando un nuovo oggetto
   - chiediti come fa quest'oggetto a soddisfare questa responsabilità (può fare tutto da solo oppure interagire con altri oggetti)
   - sulla base di: opportun: CRITERI per l'assegnazione di responsabilità

PATTERN GRASP: (General Responsibility Assignment Software Patterns)

(coppia problema soluzione) → descrivono principi fondamentali per la progettazione di oggetti e l'assegnazione di responsabilità agli oggetti → PATTERN

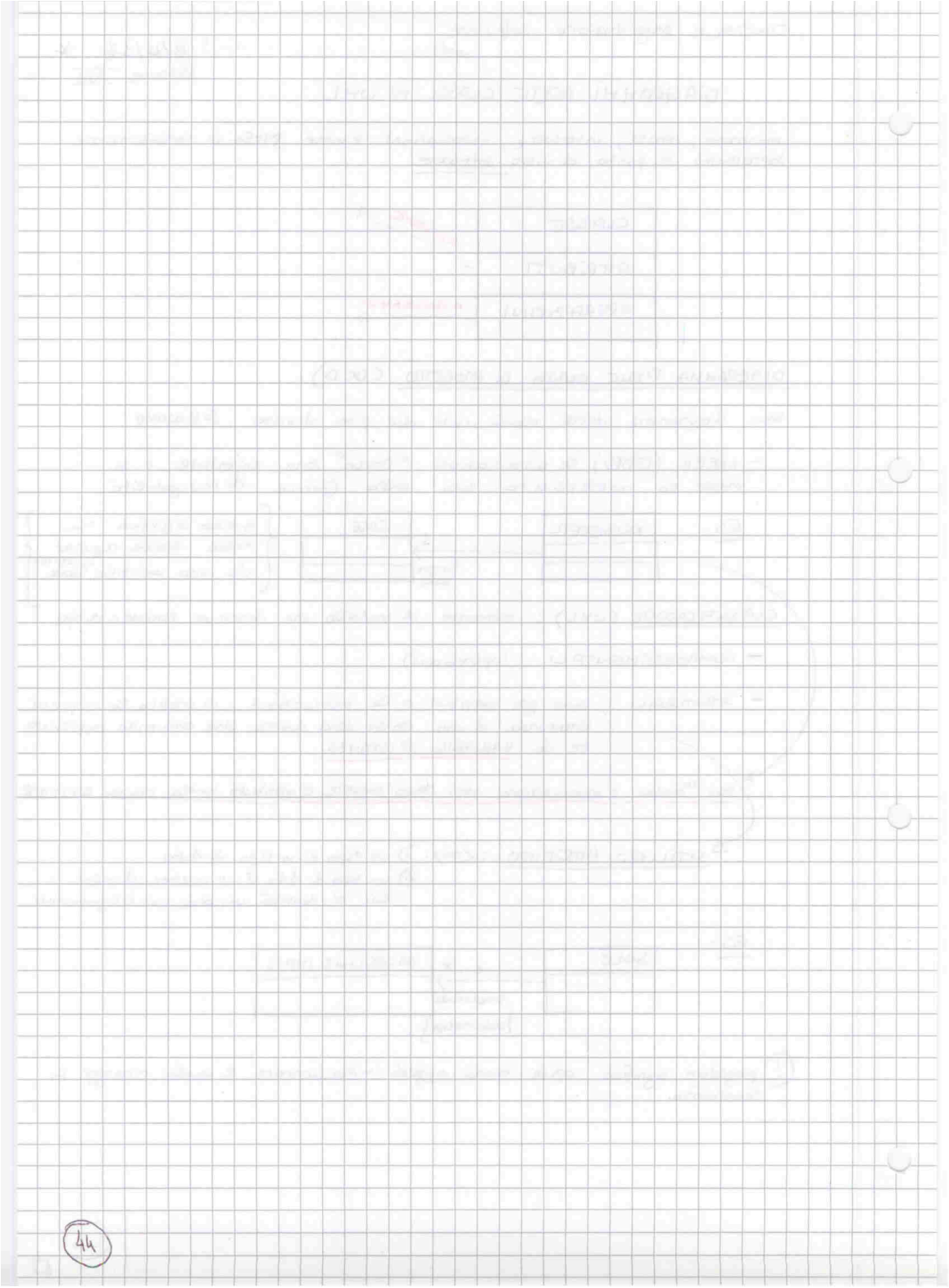
codificano criteri per l'assegnazione di responsabilità

GRASP di BASE: Creator, Information Expert, Low Coupling, High Cohesion, Controller

con un SALTO RAPPRESENTAZIONALE BASSO: distanza tra il nostro modello mentale del dominio e la rappresentazione del dominio mediante il software → codice facile da modificare

ES PATTERN: problema - soluzione

abbiamo una casa in montagna e non vogliamo che il tetto crolli → tetto a spioventi

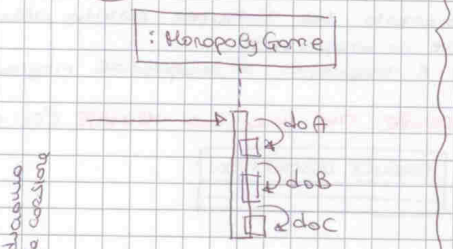

44

Analisi e progettazione software

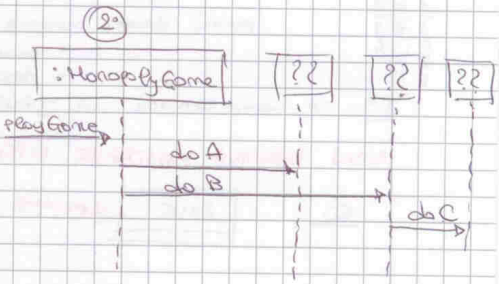
8/4/13
Lezione XIII

Controller : 2 POSSIBILITÀ

1°



2°



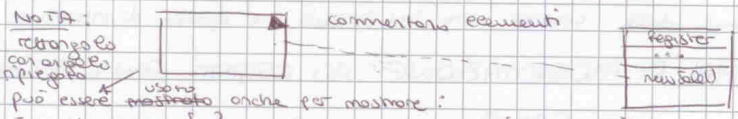
Introduzione sistema

(Quale è la migliore tra queste due?

COESIONE: è una misura di quanto lavoro viene svolto da un elemento SOFTWARE (quanto sono correlate le operazioni/responsabilità)

- HIGH COESION:
  - Problema: come mantenere gli oggetti focalizzati, comprensibili e gestibili
  - Soluzione: assegna le responsabilità in modo tale che la COESIONE rimanga alta.
    Utilizzo questo principio per valutare e confrontare le alternative

NOTE, COMMENTI, E VINCOLI: [UML]



due p.to di vista comportamentale (UML)

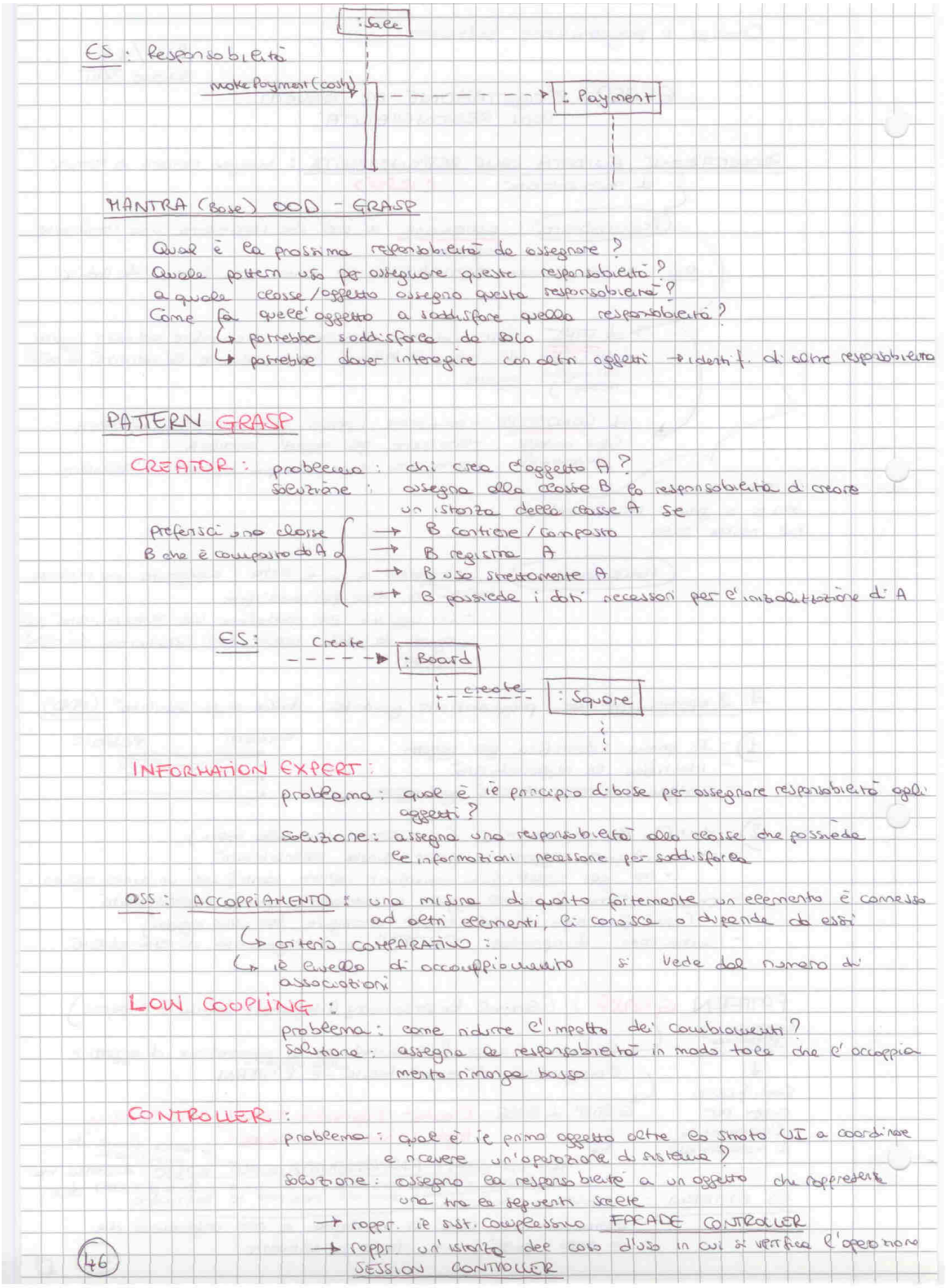
OPERAZIONE: (prototipo → interrogazione che un oggetto può essere chiamato ad eseguire)

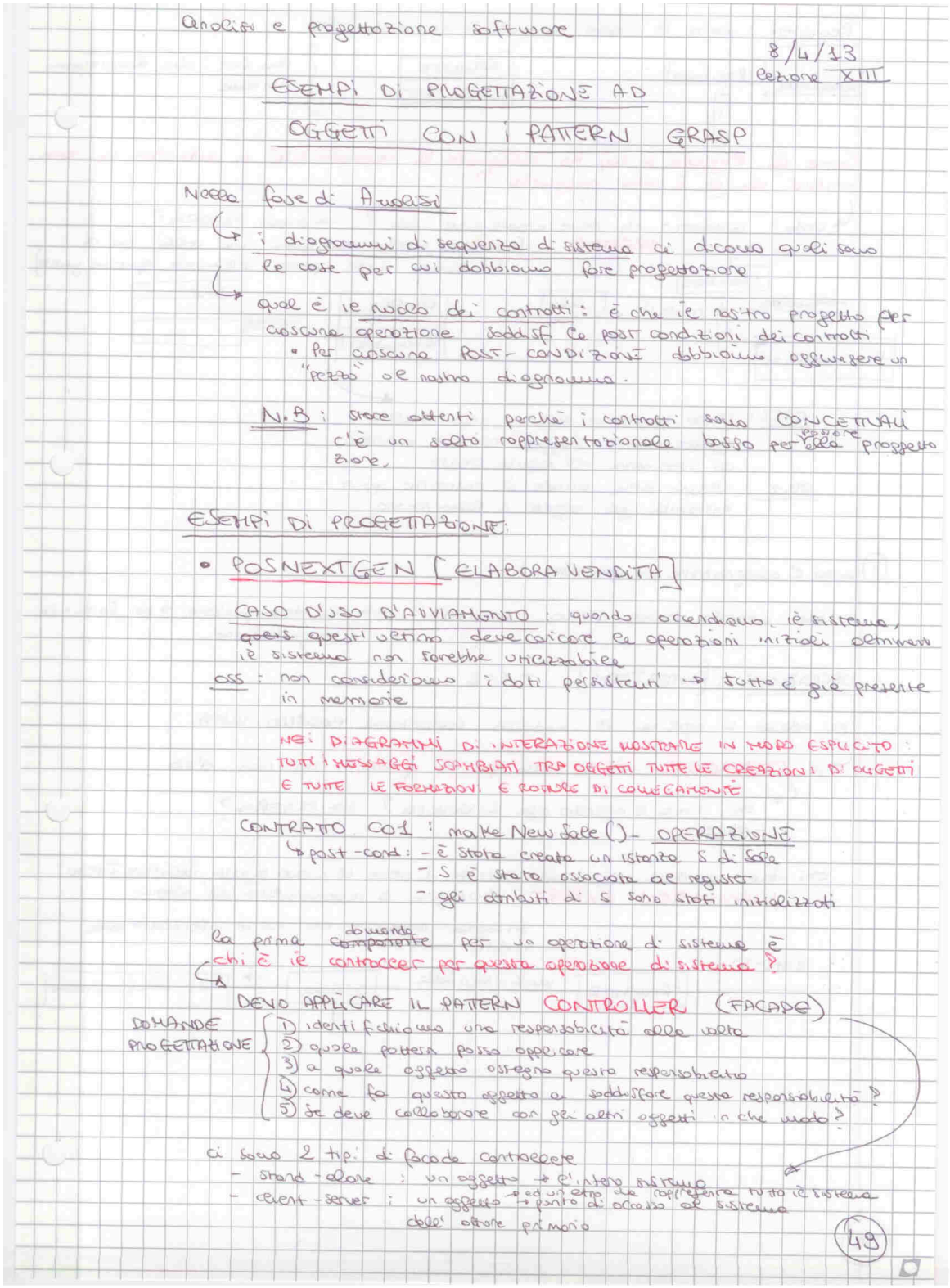
Metodo: implementazione (viene fatta con le note)

Costruttore: op. create corrisponde ad un costruttore → mostrati con lo stereotipo «constructor» ← in GENERALE per le parole CHIAVI

Vincolo: condizione booleana o restrizione {abstract} {restricted}

Proprietà: un valore con un nome, denota una caratteristica di un elemento
- VISIBILITÀ: proprietà delle operazioni


:Sale
ES: Responsabilità
makePayment(cash)
:Payment
MANTRA (Base) OOD - GRASP
Qual è la prossima responsabilità da assegnare?
Quale pattern uso per assegnare questa responsabilità?
a quale classe/oggetto assegno questa responsabilità?
Come fa quell'oggetto a soddisfare quella responsabilità?
↳ potrebbe soddisfarla da solo
↳ potrebbe dover interagire con altri oggetti → identif. di altre responsabilità
PATTERN GRASP
CREATOR: problema: chi crea l'oggetto A?
soluzione: assegna alla classe B la responsabilità di creare un istanza della classe A se
Preferisci una classe B che è composta da A
→ B contiene / compositi
→ B registra A
→ B usa strettamente A
→ B possiede i dati necessari per l'inizializzazione di A
ES:
Create
:Board
create
:Square
INFORMATION EXPERT:
problema: qual è il principio di base per assegnare responsabilità agli oggetti?
soluzione: assegna una responsabilità alla classe che possiede le informazioni necessarie per soddisfarla
OSS: ACCOPPIAMENTO: una misura di quanto fortemente un elemento è connesso ad altri elementi, li conosce o dipende da essi
↳ criterio COMPARATIVO:
↳ il livello di accoppiamento si vede dal numero di associazioni
LOW COUPLING:
problema: come ridurre l'impatto dei cambiamenti?
soluzione: assegna le responsabilità in modo tale che l'accoppiamento rimanga basso
CONTROLLER:
problema: qual è il primo oggetto oltre lo strato UI a coordinare e ricevere un'operazione di sistema?
soluzione: assegno la responsabilità a un oggetto che rappresenta uno tra le seguenti scelte
→ rappr. il sist. complessivo FACADE CONTROLLER
→ rappr. un'istanza del caso d'uso in cui si verifica l'operazione SESSION CONTROLLER
46



Analisi e progettazione software

8/4/13
Lezione XIII

ESEMPI DI PROGETTAZIONE AD OGGETTI CON I PATTERN GRASP

Nella fase di Analisi
- i diagrammi di sequenza di sistema ci dicono quali sono le cose per cui dobbiamo fare progettazione
- qual è il ruolo dei contratti: è che il nostro progetto per ciascuna operazione soddisfi le post condizioni dei contratti
  - Per ciascuna POST-CONDIZIONE dobbiamo aggiungere un "pezzo" al nostro diagramma.

N.B: stare attenti perché i contratti sono CONCETTUALI cioè un livello rappresentazionale basso per la fase di progettazione.

ESEMPI DI PROGETTAZIONE:

- POSNEXTGEN [ELABORA VENDITA]

CASO D'USO D'AVVIAMENTO: quando accendiamo il sistema, questi ultimo deve caricare le operazioni iniziali altrimenti il sistema non sarebbe utilizzabile

oss: non consideriamo i dati persistenti → tutto è già presente in memoria

NEI DIAGRAMMI DI INTERAZIONE MOSTRIAMO IN MODO ESPLICITO TUTTI I MESSAGGI SCAMBIATI TRA OGGETTI TUTTE LE CREAZIONI DI OGGETTI E TUTTE LE FORMAZIONI E ROTTURE DI COLLEGAMENTI

CONTRATTO CO1: makeNewSale() - OPERAZIONE
↳ post-cond: - è stata creata un istanza s di Sale
- s è stata associata al register
- gli attributi di s sono stati inizializzati

La prima componente dominio per un operazione di sistema è chi è il controller per questa operazione di sistema?

DEVO APPLICARE IL PATTERN CONTROLLER (FACADE)

DOMANDE PROGETTAZIONE
1) identifichiamo una responsabilità della volta
2) quale pattern posso applicare
3) a quale oggetto assegno questa responsabilità
4) come fa questo oggetto a soddisfare questa responsabilità?
5) se deve collaborare con gli altri oggetti in che modo?

ci sono 2 tipi di facade controller
- stand-alone: un oggetto → l'intero sistema
- client-server: un oggetto → ed un altro che rappresenta tutto il sistema → punto di accesso al sistema dell'attore primario

49

(une)

DIPENDENZE: un qualche elemento SOFTWARE dipende da qualche altro elemento software

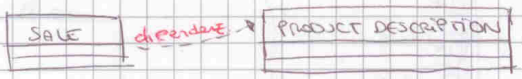
↳ MODELLAZIONE: è possibile mostrare una dipendenza mediante una freccia tratteggiata

maggiore è il numero di dipendenze maggiore è il grado di accoppiamento

perché dovrei mostrare una dipendenza?
- cambiamento del fornitore potrebbe influire sul cliente
- descrive un accoppiamento

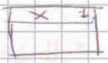
ES: la compilazione in JAVA è trasparente al compilatore programmatore

NON BISOGNA MOSTRARE DIPENDENZE CHE SONO GIÀ IMPLICITE (es: associat...)

ES: 

AGGREGAZIONE: è un associazione che rappresenta una relazione intero - parte

COMPOSIZIONE: (unez. a e.v. concettuale) è una forma forte di aggregazione in cui:
- una parte appartiene a un composto alla volta
- il composto è responsabile della creazione CANCELLAZIONE delle sue parti

CLASSE SINGLETON: è pensata per avere una sola istanza 

oss: se devo usare un oggetto di DEVE essere ordine da classe, ma se devo usare una classe non è detto che mi serva avere d'oggetto.
messaggi → operazioni

I DIAGRAMMI DI INTERAZIONE E DIAGRAMMI DELLE CLASSI VANNO FATTI IN PARALLELO

ie diagramma della classi sta UN FOGLIO e quello di interazione su un altro

ESEMPI DI PROGETTAZIONE AD OGGETTI CON i PATTERN

GRASP

in UP la progettazione prende il nome di realizzazione dei casi d'uso ed enfatizza la forte connessione tra requisiti e progetto (oggetti vanno deve soddisfare i requisiti)

→ Quando devo fare progettazione guarderò le casi d'uso dell'inizio alla fine e non a CASCATA, → devo guardare tutte le operazioni di sistema del caso d'uso

→ Per ciascuna operazione di sistema bisogna disegnare ~~un~~ (creare) un diagramma di interazione che mostra come gli oggetti della logica applicativa gestisca quella operazione di sistema

↳ nel nostro caso ciascuna delle ~~componenti~~ operazioni hanno delle componenti di trasformazioni → ~~ass~~ o interrogazioni

Analisi e progettazione del software
Anno Accademico 2012-2013
Homework 2 – Analisi – Operazioni di sistema

Regole: **SCRIVERE IN MODO LEGGIBILE E NON AMBIGUO**. Scrivere il proprio nome su ciascun foglio utilizzato, in alto a destra. Accanto allo svolgimento di ciascun esercizio o domanda va indicato chiaramente l'esercizio o la domanda a cui lo svolgimento è relativo (ad esempio, Esercizio B2, SSD per il caso d'uso UC1, oppure Esercizio B4, domanda B4.3). Alle domande in cui è prevista una risposta binaria SI/NO ("è opportuno fare questo?") è necessario scrivere in modo esplicito prima la risposta alla domanda (SI oppure NO) e poi dare una breve motivazione della risposta.

In questo homework si faccia ancora riferimento ai requisiti per il sistema **AcmeArti**, descritti in un documento separato e già utilizzati per gli homework precedenti.

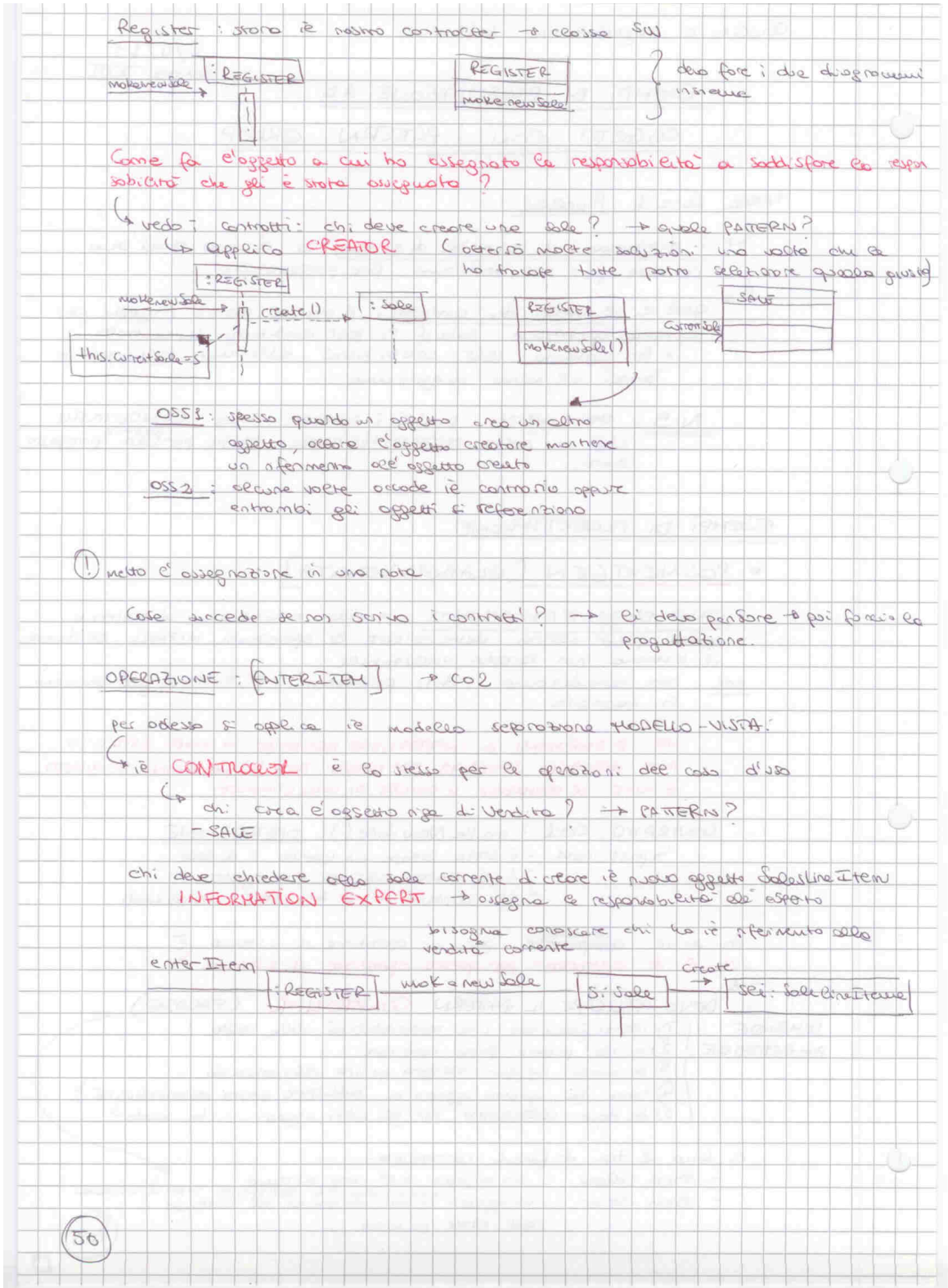
Esercizio B1 (Modellazione di dominio)
Fare l'analisi a oggetti per il sistema in discussione, come segue:
- Mostrare il modello di dominio **semplificato**, relativo a tutti casi d'uso mostrati (UC1-UC4), contenente solo classi concettuali (con il nome) e associazioni (con il nome e le molteplicità) **ma senza attributi**.

Esercizio B2 (Diagrammi di sequenza di sistema)
Fare l'analisi a oggetti per il sistema in discussione, come segue:
- Mostrare il diagramma di sequenza di sistema per lo scenario principale del caso d'uso UC1.
- Mostrare il diagramma di sequenza di sistema per lo scenario principale del caso d'uso UC2.

Esercizio B3 (Contratti delle operazioni)
Fare l'analisi a oggetti per il sistema in discussione, come segue:
- Mostrare i contratti di tutte le operazioni di sistema per il caso d'uso UC2 (*Compilazione Piano formativo individuale*), compresa l'operazione corrispondente all'estensione 4-7a.

Esercizio B4 (Domande)
Relativamente al precedente esercizio B2, rispondere alle seguenti domande (rispondere in modo sintetico, scrivendo circa 3-5 righe per ciascuna domanda):
- (B4.1) Relativamente al caso d'uso UC2, è opportuno mostrare un evento/operazione di sistema in corrispondenza al solo passo 1 del caso d'uso? Ed in corrispondenza al solo passo 9 del caso d'uso? Motivare la risposta, discutendo anche l'eventuale effetto provocato dall'esecuzione di tali operazioni di sistema.
- (B4.2) Relativamente al caso d'uso UC2, è opportuno chiamare "inserisciCodiceEPassword" l'evento/operazione di sistema che può essere identificato in corrispondenza al passo 2 del caso d'uso? Motivare la risposta, proponendo eventualmente un nome più opportuno per tale evento di sistema.
- (B4.3) Relativamente al caso d'uso UC2, è opportuno introdurre un evento/operazione di sistema in corrispondenza al passo 3 del caso d'uso? Motivare la risposta, discutendo anche l'eventuale effetto provocato dall'esecuzione di una tale operazione di sistema.
- (B4.4) Relativamente al passo 8 del caso d'uso UC2, discutere l'impatto della frase "genera una ricevuta" sul diagramma di sequenza di sistema per tale caso d'uso.

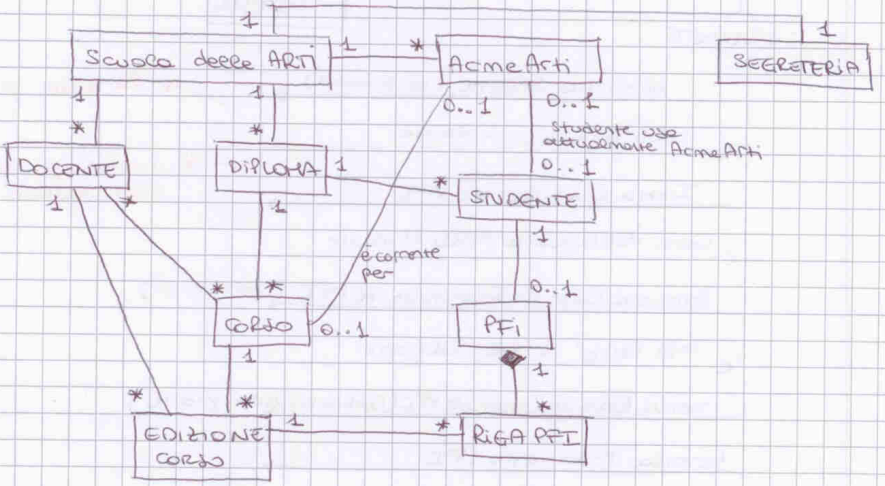
Esercizio B5 (Domande)
Relativamente al precedente esercizio B3, rispondere alle seguenti domande (rispondere in modo sintetico, scrivendo circa 3-5 righe per ciascuna domanda):
- (B5.1) Relativamente ai contratti delle operazioni di sistema per il caso d'uso UC2, è opportuno introdurre in uno di tali contratti la post-condizione "è stato creato un nuovo Studente s"? Se sì, in quale operazione? Motivare la risposta.
- (B5.2) Relativamente ai contratti delle operazioni di sistema per il caso d'uso UC2, è opportuno introdurre in uno di tali contratti la post-condizione "è stato creato un nuovo Piano formativo individuale pfi"? Se sì, in quale operazione? Motivare la risposta.
- (B5.3) Relativamente ai contratti delle operazioni di sistema per il caso d'uso UC2, è opportuno introdurre in uno di tali contratti la post-condizione "sono state mostrate informazioni sulle edizioni previste per il corso scelto"? Se sì, in quale operazione? Motivare la risposta.
- (B5.4) Relativamente ai contratti delle operazioni di sistema per il caso d'uso UC2, è più opportuno introdurre in uno di tali contratti la post-condizione "il Corso c è stato aggiunto alla lista dei corsi scelti dallo Studente s" oppure la post-condizione "il Corso c è stato aggiunto al Piano formativo individuale pfi dello Studente s"? Se sì, in quale operazione? Motivare la risposta.


Register : store è nostro controller → classe SW
: REGISTER
makeNewSale
REGISTER
makeNewSale()
devo fare i due diagrammi insieme
Come fa l'oggetto a cui ho assegnato la responsabilità a soddisfare la responsabilità che gli è stata assegnata ?
vedo i contratti: chi deve creare una sale ? → quale PATTERN ?
Applico CREATOR (ottengo molte soluzioni una volta che le ho trovate tutte parto selezionare quella giusta)
: REGISTER
makeNewSale
create()
: Sale
this.currentSale = s
REGISTER
makeNewSale()
Corrisponde
SALE
OSS1: spesso quando un oggetto crea un altro oggetto, allora l'oggetto creatore mantiene un riferimento all'oggetto creato
OSS2: alcune volte accade il contrario oppure entrambi gli oggetti si referenziano
① netto è assegnazione in una note
Cosa succede se non servo i contratti ? → ci devo pensare poi fa parte la progettazione.
OPERAZIONE : enterItem → CO2
per questo si applica il modello separazione MODELLO-VISTA.
il CONTROLLER è lo stesso per le operazioni del caso d'uso
chi crea l'oggetto riga di vendita ? → PATTERN ?
- SALE
chi deve chiedere alla sale corrente di creare il nuovo oggetto SaleLineItem
INFORMATION EXPERT → assegna la responsabilità all'esperto
bisogna conoscere chi ha il riferimento alla vendita corrente
enterItem
: REGISTER
makeNewSale
5: Sale
create
sli: SaleLineItem

Analisi e progettazione software

11/4/13
Lezione XIV

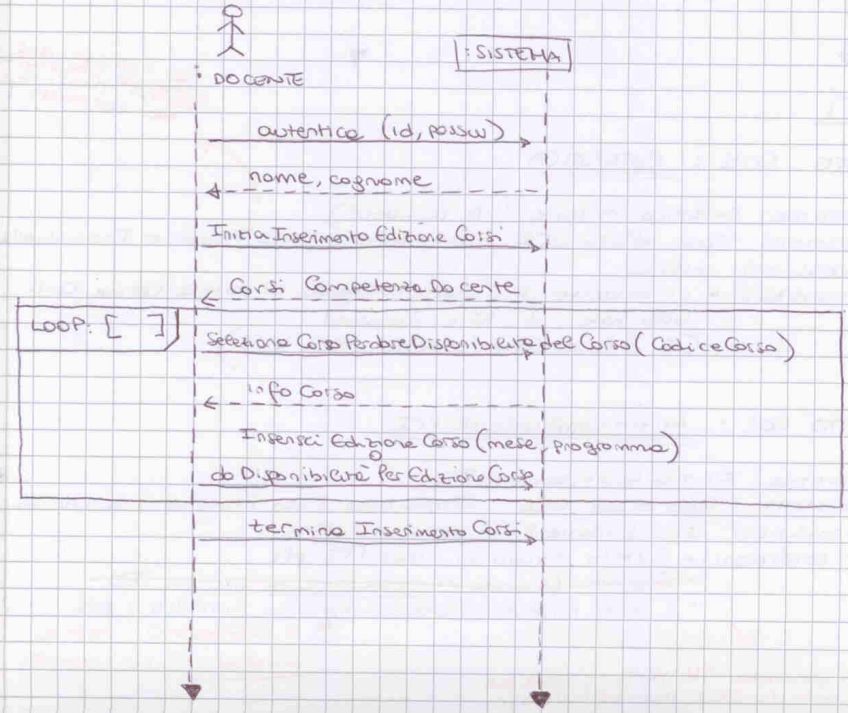
HOMEWORK 2 : [ANALISI - OPERAZIONI DI SISTEMA]



ASSOCIAZIONI → implicano la necessità di ricordare NON AZIONI ma FATTI

OSS: risposta convenzionale : è ie banchetto che si ricorda chi si è autenticato

DIAGR. DI SEQUENZA DI SISTEMA : [SSD]



Analisi e progettazione software

11/4/13
Lezione XIV

CONTRATTI

CONTRATTO CO3 : SELEZIONA CORSO PER INSERIMENTO PFI

- Operazione: SelezionaCorsoPerInserimentoNeiPFI (codiceCorso)
- Riferimenti: Caso d'uso UC2 : Compilazione PIANO formativo Individuale
- Pre-condizioni: uno studente s sta usando un sistema e sta compilando ie proprio pfi
- Post-condizione: ~~• è stato formato un collegamento tra portale Acme Arti e un corso c, sulla base di codice Corso.~~

CONTRATTO CO4 : INSERISCI EDIZIONE CORSO NEL PFI

- Operazione: InserisciEdizioneCorsoNeiPFI (codEdizione, priorità)
- Pre-condizioni: uno Studente s sta inserendo un PFI pfi ~~e ha selezionato ie corso~~ c
- Riferimento: Caso d'uso UC2 Compilazione PIANO formativo Individ
- Post-condizione: • È stato creato una nuova RIGA PFI r
  • È stato formato un collegamento tra pfi ed r
  • È stato formato un collegamento tra r e un Edizione Corso ec sulla base di codEdizione
  • R.priorità è diventato priorità

CONTRATTO CO5 : TERMINA INSERIMENTO PFI ()

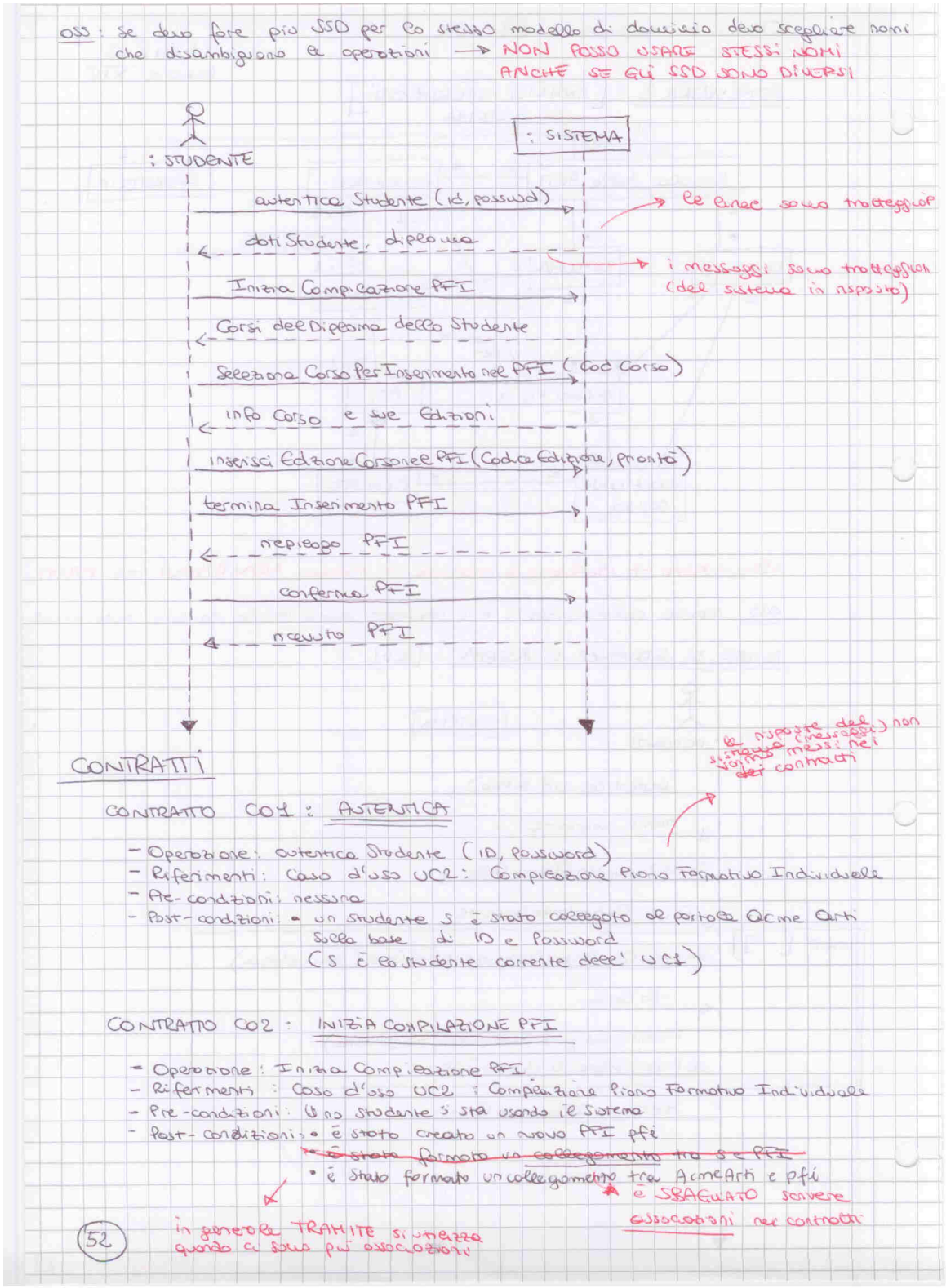
- Operazione: terminaInserimentoPFI()
- Riferimento: Caso d'Uso UC2 : Compilazione PIANO FORMATIVO INDIVID
- Pre-condizioni: uno Studente s sta inserendo un PFI pfi
- Post-condizioni: • Nessuno.

CONTRATTO CO6 : CONFERMA INSERIMENTO PFI()

- Operazione: confermaInserimentoPFI()
- Riferimenti: caso d'uso UC2: Compilazione Piano Formativo Individuale
- Pre-condizioni: uno studente s sta inserendo un PFI pfi
- Post-condizioni: • è stato creato un nuovo PFI pfi
  • ~~è stato formato un collegamento tra lo studente s e ie PFI pfi~~
  • È stato formato un collegamento tra AcmeArti e pfi
  • È stato formato un collegamento tra lo studente s e ie PFI pfi

CONTRATTO CO7 : ANNULLA INSERIMENTO PFI

- Operazione: annullaInserimentoPFI()
- Riferimenti: Caso d'uso UC2: compilazione Piano formativo Individuale
- Pre-condizioni: uno Studente s sta inserendo un PFI pfi
- Post-condizioni: • sono state distrutte tutte le righe PFI collegate a pfi
  • sono stati rotti i collegamenti tra queste righe PFI e pfi
  • sono stati rotti i collegamenti tra queste righe PFI e le relative EdizioniCorso
  • È stato rotto ie collegamento tra pfi ed s
  • È stato distrutto ie PFI pfi


OSS: se devo fare più SSD per lo stesso modello di dominio devo scegliere nomi che disambiguano le operazioni → NON POSSO USARE STESSI NOMI ANCHE SE GLI SSD SONO DIVERSI
: STUDENTE
: SISTEMA
autentica Studente (id, password)
dati Studente, dipendenza
Inizia Compilazione PFI
Corsi del Dipartimento dello Studente
Seleziona Corso Per Inserimento nel PFI (cod Corso)
info Corso e sue Edizioni
inserisci Edizione Corso nel PFI (Codice Edizione, priorità)
termina Inserimento PFI
riepilogo PFI
conferma PFI
ricevuto PFI
le enec sono tratteggiate
i messaggi sono tratteggiati (del sistema in risposta)
CONTRATTI
CONTRATTO CO1: AUTENTICA
- Operazione: autentica Studente (ID, password)
- Riferimenti: Caso d'uso UC2: Compilazione Piano Formativo Individuale
- Pre-condizioni: nessuna
- Post-condizioni: • un studente s è stato collegato al portale Acme Arti sulla base di ID e Password
(s è lo studente corrente dell'UC1)
CONTRATTO CO2: INIZIA COMPILAZIONE PFI
- Operazione: Inizia Compilazione PFI
- Riferimenti: Caso d'uso UC2: Compilazione Piano Formativo Individuale
- Pre-condizioni: Uno studente s sta usando il sistema
- Post-condizioni: • è stato creato un nuovo PFI pfi
• è stato formato un collegamento tra AcmeArti e pfi
è SBAGLIATO scrivere associazioni nei contratti
in generale TRAMITE si intende quando ci sono più associazioni
52

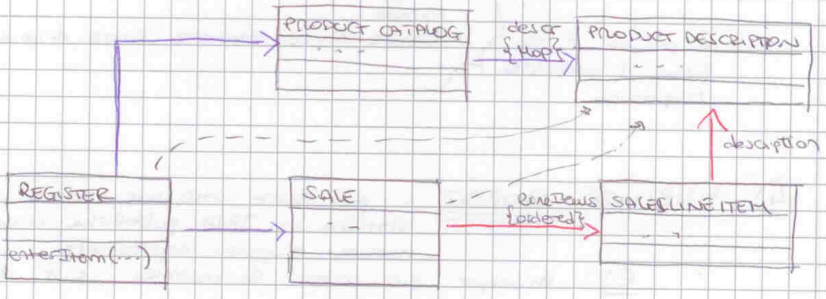
Analisi e progettazione software

15/4/13
lezione XV

ENTERITEM → è l'operazione più frequente, quindi: posso pensare che le register conosca per tutto la giornata il catalogo prodotto/i naturalmente ci dovrà essere la visibilità adeguata

le operazioni vanno create in base alla priorità descritta dal diagramma di interazione.

CONVERSIONE: da tipo semplice ad OGGETTO viene effettuata attraverso una mappa (soluzione comune)



- navigare i collegamenti che può devono essere stati creati da qualcuno
- importanti per formare altri collegamenti

allo fine bisogna verificare che ogni collegamento progettato sia utile e che venga effettivamente formato

PRODUCT DESCRIPTION → nella versione finale sarà trovata e caricata da una base di dati

SUPPOSIZIONE: per ora tutti gli oggetti di cui le informazioni sono disponibili vengono creati e caricati in memoria durante il caso d'uso d'avviamento

PROGETTARE PER LA VISIBILITÀ

17/4/13
lezione XVI

un oggetto (mittente) può inviare un messaggio a un oggetto (destinatario) solo se il mittente ha visibilità del destinatario
- NELL'INVIO DI UN MESSAGGIO, IL MITTENTE DEVE CONOSCERE UN RIFERIMENTO AL DESTINATARIO -

! il progetto DEVE garantire tutte le visibilità tra oggetti necessarie per consentire l'interazione tra oggetti.

VISIBILITÀ: è la capacità di un oggetto di vedere o avere un riferimento ad un altro oggetto → hanno DURATA temporale diversa

1) VISIBILITÀ PER ATTRIBUTO: quando quell'oggetto ha una variabile di istanza che referenzia un altro oggetto.

più importante perché vive anche dopo la vita del metodo

- è una forma di visibilità relativamente duratura (sopravvive all'esecuzione dei metodi istanza)

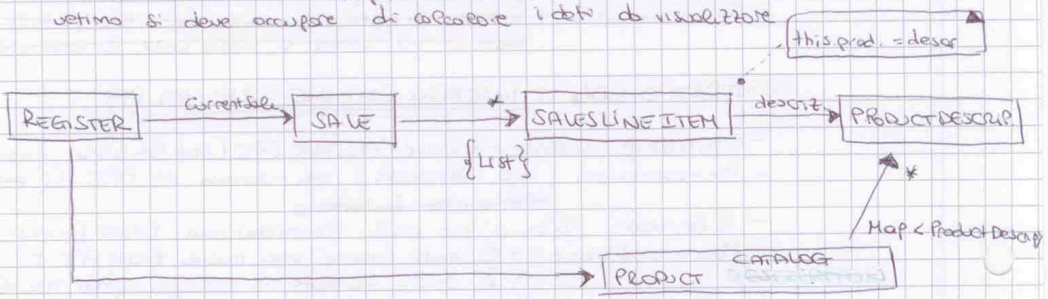
PROGETTAZIONE:

15/4/13
lezione XV

- chiediti quale è la responsabilità da assegnare
- sulla base del tipo di responsabilità, selezione le pattern GRASP da applicare
- sulla base dei PATTERN, scegli l'oggetto a cui assegnare la responsabilità
- chiediti come fa l'oggetto scelto a soddisfare la responsabilità
  → questo può portare alla identificazione di altre responsabilità da assegnare

Bisogna progettare per la visualizzazione dei dati?

visualizzare dati non è responsabilità della stato di dominio, ma bensì quest'ultimo si deve occupare di calcolare i dati da visualizzare



La prima cosa da fare dopo aver scelto l'attività di analisi è quella di capire il verso di navigabilità.

Le associazioni a 1 sono rappresentate da una variabile di istanza semplice
Le associazioni a molti sono rappresentati da una variabile d'istanza di tipo:
ad esempio una LISTA

formare un nuovo collegamento significa fare un add alla collezione oppure un assegnazione alla variabile d'istanza
chi crea l'oggetto collezione? → verrà creata dalla vendita

quando creo l'oggetto collezione? → verrà creata quando creo la vendita

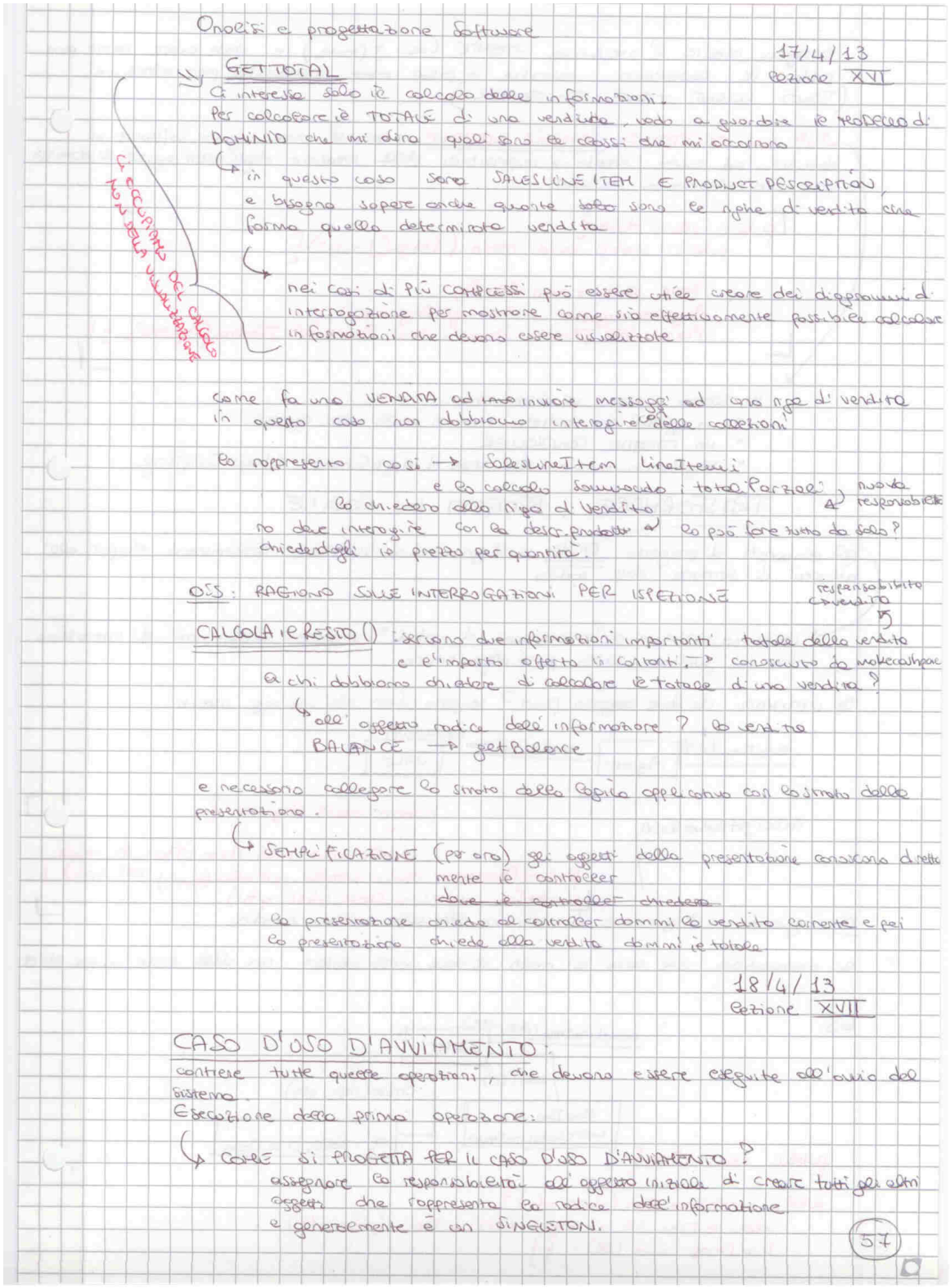
OSS: in certi casi l'unico modo per capire se l'operazione di sistema è giusta è andare avanti e vedere casi analoghi (ragionare)

→ chi esegue la PRESENTAZIONE vive in un processo diverso dal processo della logica applicativa → non si possono scambiare oggetti

Nel modello di dominio tutte le istanze di una classe non possono essere rappresentati da una classe (una classe può rappresentare un'istanza) → vengono invece rappresentate da un'associazione

Come si rappresenta in un DCD (diagramma delle classi di progetto) una collezione di tipo MAP <TIPO BASE, OGGETTO> non è mai un tipo di base

va dato l'oggetto che detiene la mappa dell'oggetto che contiene "OGGETTO"
è per convenzione si ometterà di scrivere il tipo base e sopra la descrizione scriverò Map < ProductDescription >



Analisi e progettazione Software

17/4/13
Lezione XVI

GET TOTAL

Ci interessa solo il calcolo delle informazioni.
Per calcolare il TOTALE di una vendita, vado a guardare il modello di DOMINIO che mi dice quali sono le classi che mi occorrono

↳ in questo caso sarà SALESLINEITEM e PRODUCTDESCRIPTION, e bisogna sapere anche quante sono le righe di vendita che formano quella determinata vendita

↳ nei casi di più complessi può essere utile creare dei diagrammi di interrogazione per mostrare come sia effettivamente possibile calcolare informazioni che devono essere visualizzate

Ci occupiamo del calcolo non della visualizzazione

come fa una VENDITA ad inviare messaggi ad una riga di vendita in questo caso noi dobbiamo interrogare degli oggetti

Lo rappresento così → SaleLineItem LineItems
e lo calcolo sommando: totale Parziale → nuova responsabilità
lo chiedo alla riga di vendita
no deve interrogare con lo descr. prodotto → lo può fare tutto da solo?
chiedendogli il prezzo per quantità.

OSS: RAGIONO SULLE INTERROGAZIONI PER ISPEZIONE

responsabilità conferita

CALCOLA IL RESTO(): servono due informazioni importanti: totale della vendita e l'importo offerto in contanti. → conosciuto da MoneyExchange

a chi dobbiamo chiedere di calcolare il totale di una vendita?

↳ alla oggetto radice delle informazioni ? la vendita
BALANCE → getBalance

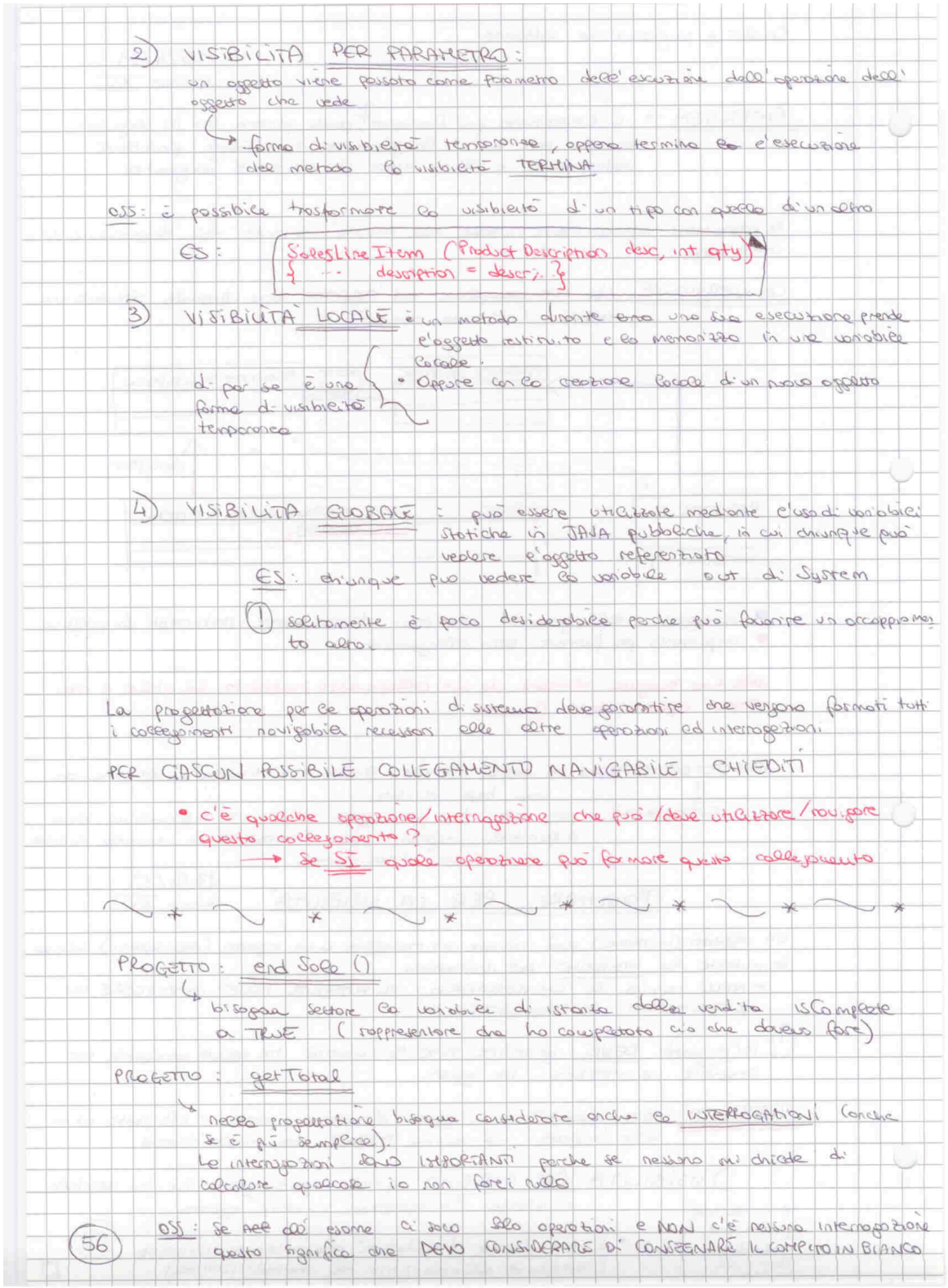
e necessario collegare lo strato della logica applicativa con lo strato della presentazione.

↳ SEMPLIFICAZIONE (per ora) gli oggetti della presentazione conoscono direttamente il controller
~~dove il controller chiedere~~
la presentazione chiede al controller dominio la vendita corrente e poi la presentazione chiede alla vendita dominio il totale

18/4/13
Lezione XVII

CASO D'USO D'AVVIAMENTO:
contiene tutte queste operazioni, che devono essere eseguite all'avvio del sistema.
Esecuzione della prima operazione:

↳ come si PROGETTA PER IL CASO D'USO D'AVVIAMENTO ?
assegnare la responsabilità all'oggetto iniziale di creare tutti gli altri oggetti che rappresenta la radice dell'informazione
e generalmente è un SINGLETON.



2) VISIBILITÀ PER PARAMETRO:
un oggetto viene passato come parametro dell'esecuzione dell'operazione dell'oggetto che vede
→ forma di visibilità temporanea, oppure termina se è l'esecuzione del metodo la visibilità TERMINA

OSS: è possibile trasformare la visibilità di un tipo con quella di un altro

ES:
```
SaleLineItem (ProductDescription desc, int qty)
{ ... description = desc; }
```

3) VISIBILITÀ LOCALE: un metodo durante una sua esecuzione prende l'oggetto restituito e lo memorizza in una variabile locale.
- Oppure con la creazione locale di un nuovo oggetto

di per se è una forma di visibilità temporanea

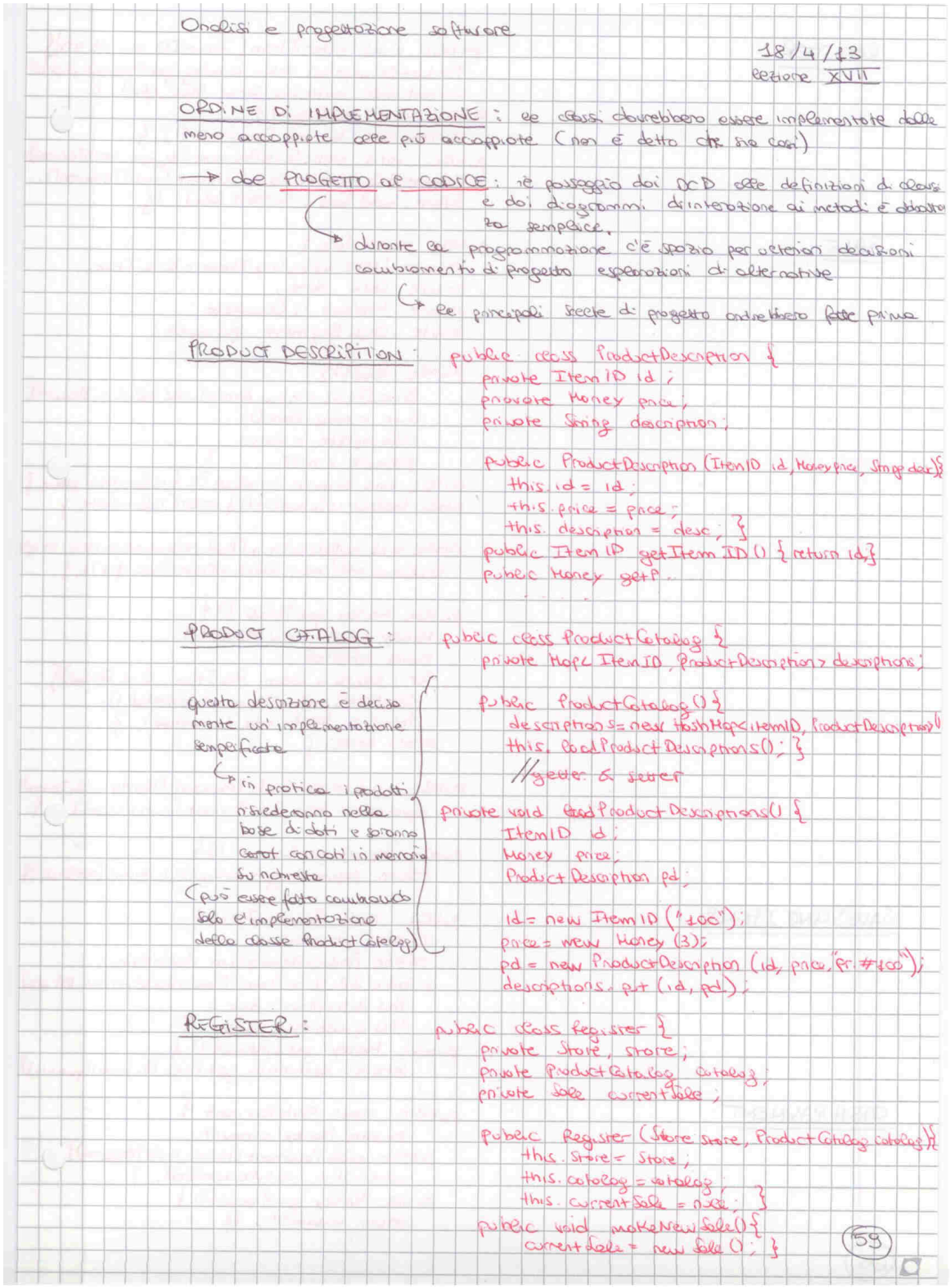
4) VISIBILITÀ GLOBALE: può essere utilizzata mediante l'uso di variabili statiche in JAVA pubbliche, in cui chiunque può vedere l'oggetto referenziato
ES: chiunque può vedere la variabile out di System
(!) solitamente è poco desiderabile perché può favorire un accoppiamento alto

La progettazione per le operazioni di sistema deve garantire che vengano formati tutti i collegamenti navigabili necessari alle altre operazioni ed interrogazioni.
PER CIASCUN POSSIBILE COLLEGAMENTO NAVIGABILE CHIEDITI
- c'è qualche operazione/interrogazione che può/deve utilizzare/rompere questo collegamento?
→ se SI quale operazione può formare questo collegamento

PROGETTO: endSale()
↳ bisogna settare la variabile di istanza della vendita isComplete a TRUE (rappresentare che ho completato ciò che dovevo fare)

PROGETTO: getTotal
↳ nella progettazione bisogna considerare anche le INTERROGAZIONI (anche se è più semplice).
Le interrogazioni SONO IMPORTANTI perché se nessuno mi chiede di calcolare qualcosa io non farei nulla

OSS: se nel DCD esiste ci sono solo operazioni e NON c'è nessuna interrogazione questo significa che DEVO CONSIDERARE DI CONSEGNARE IL COMPITO IN BIANCO



Analisi e progettazione software

18/4/13
lezione XVII

ORDINE DI IMPLEMENTAZIONE: le classi dovrebbero essere implementate dalle meno accoppiate alle più accoppiate (non è detto che sia così)

→ del PROGETTO DEL CODICE: il passaggio dai DCD alle definizioni di classe e dai diagrammi di interazione ai metodi è abbastanza semplice.

durante la progettazione c'è spazio per ulteriori decisioni, cambiamenti di progetto, esplorazioni di alternative

↳ le principali scelte di progetto andrebbero fatte prima

PRODUCT DESCRIPTION:

```
public class ProductDescription {
    private ItemID id;
    private Money price;
    private String description;

    public ProductDescription(ItemID id, Money price, String desc){
        this.id = id;
        this.price = price;
        this.description = desc; }
    public ItemID getItemID() { return id; }
    public Money getP...
```

PRODUCT CATALOG:

```
public class ProductCatalog {
    private Map<ItemID, ProductDescription> descriptions;

    public ProductCatalog() {
        descriptions = new HashMap<ItemID, ProductDescription>();
        this.fakeProductDescriptions(); }
    //getter & setter
    private void fakeProductDescriptions() {
        ItemID id;
        Money price;
        ProductDescription pd;

        id = new ItemID("100");
        price = new Money(3);
        pd = new ProductDescription(id, price, "pr #100");
        descriptions.put(id, pd);
```

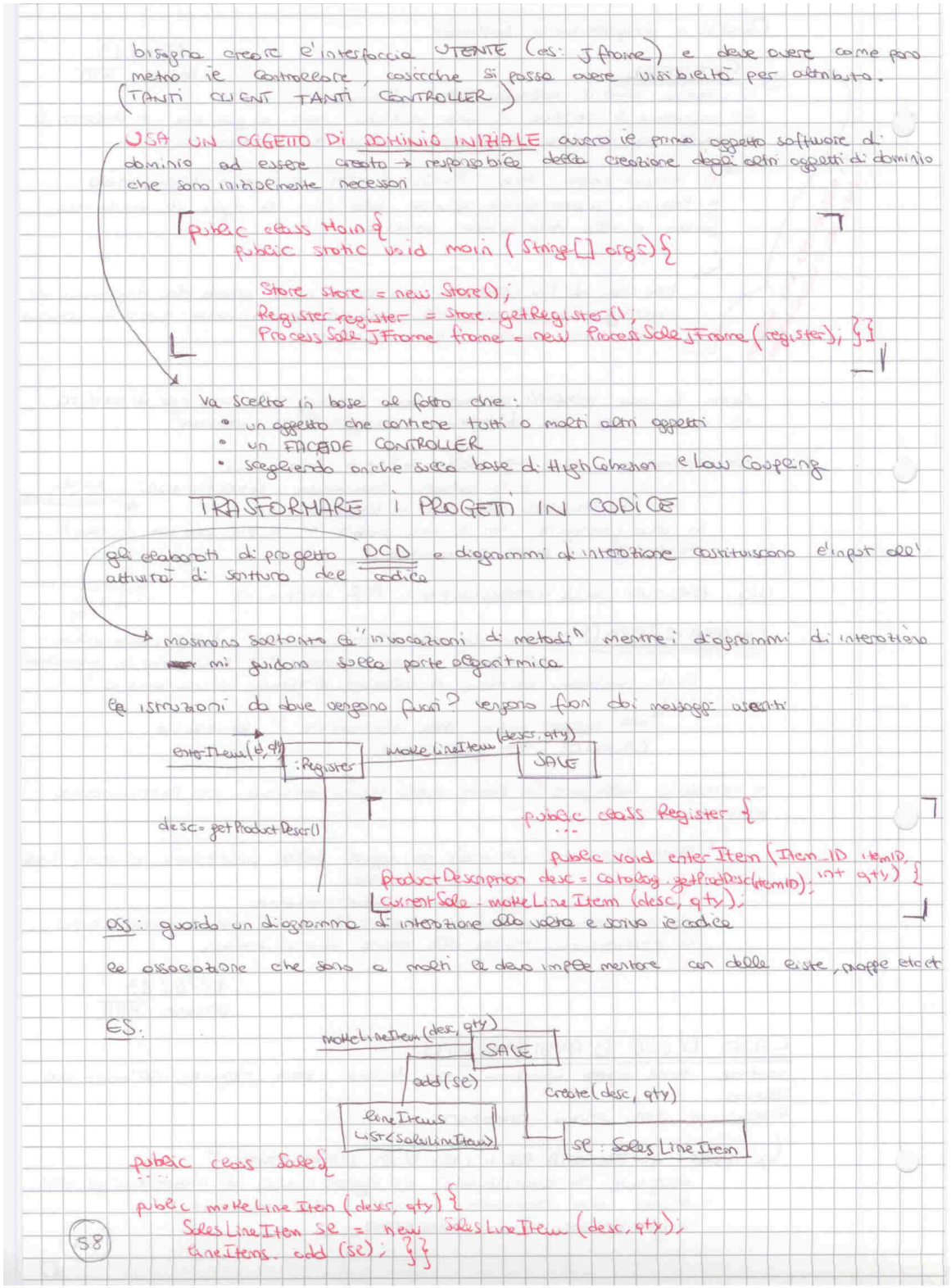
questa descrizione è decisamente un'implementazione semplificata

↳ in pratica i prodotti risiederanno nella base di dati e saranno caricati in memoria su richiesta

(può essere fatto cambiando solo l'implementazione della classe ProductCatalog)

REGISTER:

```
public class Register {
    private Store store;
    private ProductCatalog catalog;
    private Sale currentSale;

    public Register(Store store, ProductCatalog catalog){
        this.store = store;
        this.catalog = catalog;
        this.currentSale = null; }
    public void makeNewSale(){
        currentSale = new Sale(); }
```


bisogna creare l'interfaccia UTENTE (es: JFrame) e deve avere come parametro il Controllore, cosicché si passa avere visibilità per attributo.
(TANTI CLIENT TANTI CONTROLLER)
USA UN OGGETTO DI DOMINIO INIZIALE ovvero il primo oggetto software di dominio ad essere creato → responsabile della creazione degli altri oggetti di dominio che sono inizialmente necessari
public class Main {
public static void main (String[] args) {
Store store = new Store();
Register register = store.getRegister();
ProcessSaleJFrame frame = new ProcessSaleJFrame (register); }}
Va scelto in base al fatto che:
un oggetto che contiene tutti o molti altri oggetti
un FACADE CONTROLLER
scegliendo anche sulla base di: HighCohesion e Low Coupling
TRASFORMARE I PROGETTI IN CODICE
gli elaborati di progetto DCD e diagrammi di interazione costituiscono l'input dell'attività di scrittura del codice
mostrano sostanzialmente le "invocazioni di metodi" mentre i diagrammi di interazione mi guidano sulla parte algoritmica
le istruzioni da dove vengono fuori? vengono fuori dai messaggi usciti
enterItem(id, qty)
:Register
makeLineItem(desc, qty)
SALE
desc = getProductDesc()
public class Register {
public void enterItem (ItemID itemID, int qty) {
ProductDescription desc = catalog.getProdDesc(itemID);
CurrentSale.makeLineItem(desc, qty);
PS: guardo un diagramma di interazione alla volta e scrivo il codice
le associazioni che sono a molti le devo implementare con delle liste, mappe etc...
ES:
makeLineItem(desc, qty)
SALE
add(se)
create(desc, qty)
lineItems
List<SalesLineItem>
se : SalesLineItem
public class Sale {
public makeLineItem (desc, qty) {
SalesLineItem sl = new SalesLineItem (desc, qty);
lineItems.add(sl); }}
58

Analisi e progettazione Software

18/4/13
Lezione XVII

STORE:

```
public class Store {
    private ProductCatalog catalog;
    private Register register;
    private List<Sale> completedSales;

    public Store() {
        this.catalog = new ProductCatalog();
        this.register = new Register(this, catalog);
        this.completedSales = new ArrayList<Sale>();
    }
    public Register getRegister() { return register; }
    public void addSale(Sale s) {
        this.completedSales.add(s); }}
```

DIAGRAMMI STATICI → Struttura del codice
DIAGRAMMI INTERAZIONE → Istruzioni (UML)

1) è davvero utile lavorare in questo modo? Sì

2) siamo partiti dallo sviluppo iterativo → questo metodo di cui parla veramente ad abbracciare il cambiamento? Sì

* quali sono le alternative?
prima della OOP era presente della programmazione IMPERATIVA dove gli oggetti NON esistevano, ma esistevano solo le procedure e i moduli (ES: C)

MODULO per caso d'uso
PROCEDURE per ciascuna operazione di sistema

questo modo di ragionare ha il seguente pregio: ho una visione d'insieme più completa perché il codice è scritta tutta sequenziale, cosa che non succede nella programmazione OO che spezzetta tutto in oggetti
SVANTAGGI: tuttavia una volta che devo fare un cambiamento devo manipolare tantissimo codice, e dona caratteristiche da un accoppiamento alto ed una coesione bassa. (SCARSA MODIFICABILITÀ)

→ In OOP manca la visione complessiva, quindi se guardo il codice non ce la potrò mai avere.
Questo modo di visualizzazione d'insieme c'è l'ho nel diagramma di interazione.

→ per tutto quello detto in precedenza ed in più perché è presente la REFACTORING

5 PATTERN GRASP

Le cose più importante è la MODIFICABILITÀ, fare cose nuove in poco tempo
COMPRENSIBILITÀ
SEMPLICITÀ del progetto
RIUSABILITÀ

da dove vengono questi principi?
COESIONE, ACCOPPIAMENTO

6-2

REGISTER :

```
public void enterItem(ItemID id, int qty){
  ProductDescription desc = catalog.getProductDescription(
  currentSale.makeLineItem(desc, qty); }
public void endSale() {
  currentSale.becomeComplete(); }
public void makeCashPayment(Money cashTendered){
  currentSale.makeCashPayment(cashTendered);
  Store.addSale(currentSale); }}
```

SALE :

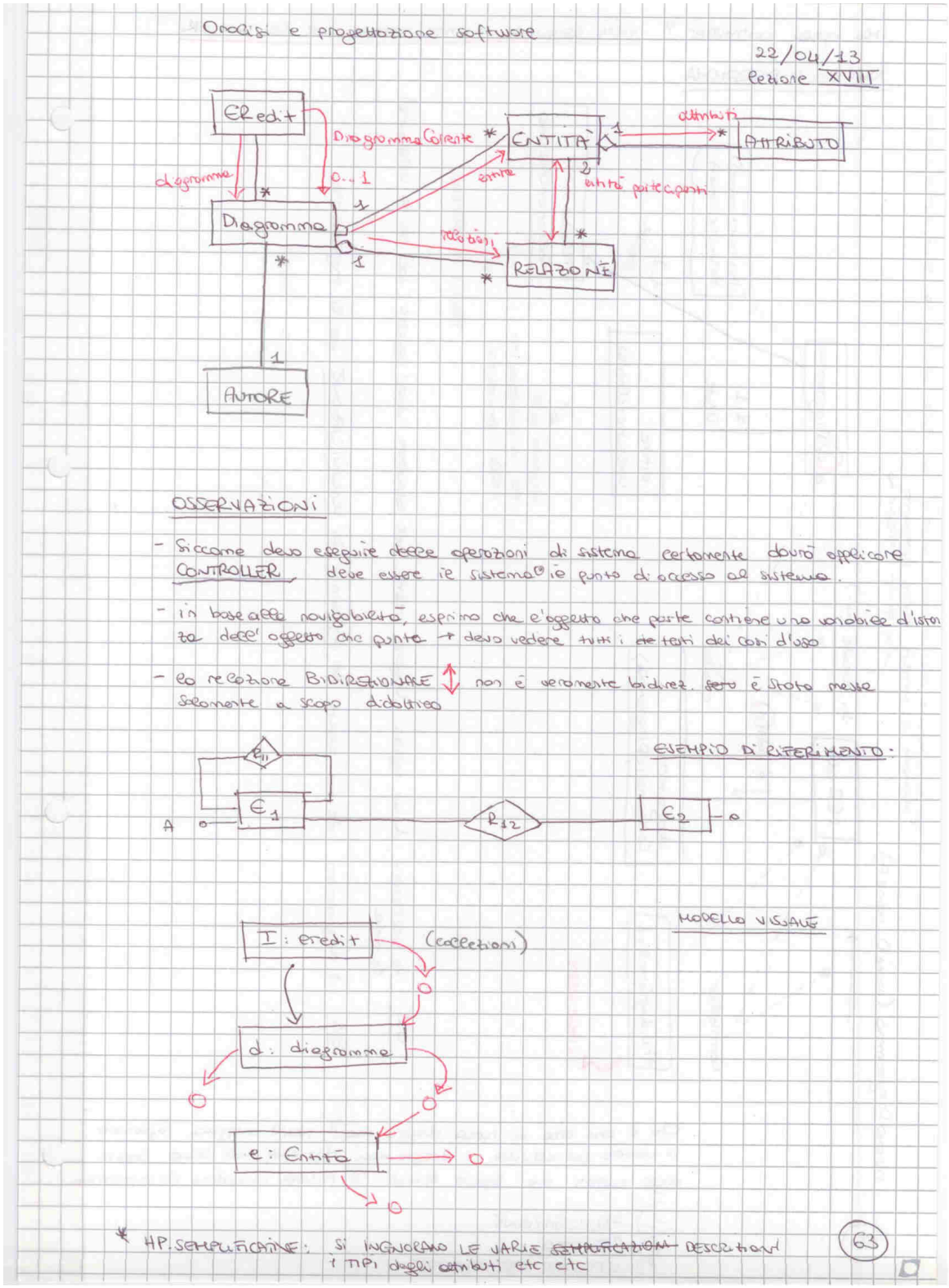
```
public class Sale {
  private List<SalesLineItem> lineItems;
  private Date date;
  private boolean isComplete;
  private CashPayment payment;

  public Sale(){
      lineItems = new ArrayList<SalesLineItem>();
      date = new Date();
      isComplete = false;
      payment = null; }
  public void becomeComplete() { isComplete = true; }
  public boolean isComplete() { return isComplete; }

  public void makeLineItem(ProductDescription desc, int qty
    lineItems.add(new SalesLineItem(desc, qty)); }

  public Money getTotal(){
    Money total = new Money(0);
    Money subtotal;
    for(SalesLineItem lineItem : lineItems){
      subtotal = lineItem.getSubtotal();
      total = total.add(subtotal); }
    return total; }

  public void makeCashPayment(Money cashTendered)
    payment = new CashPayment(cashTendered); }

  public Money getBalance(){
    return payment.getAmount().minus(getTotal()); }
```

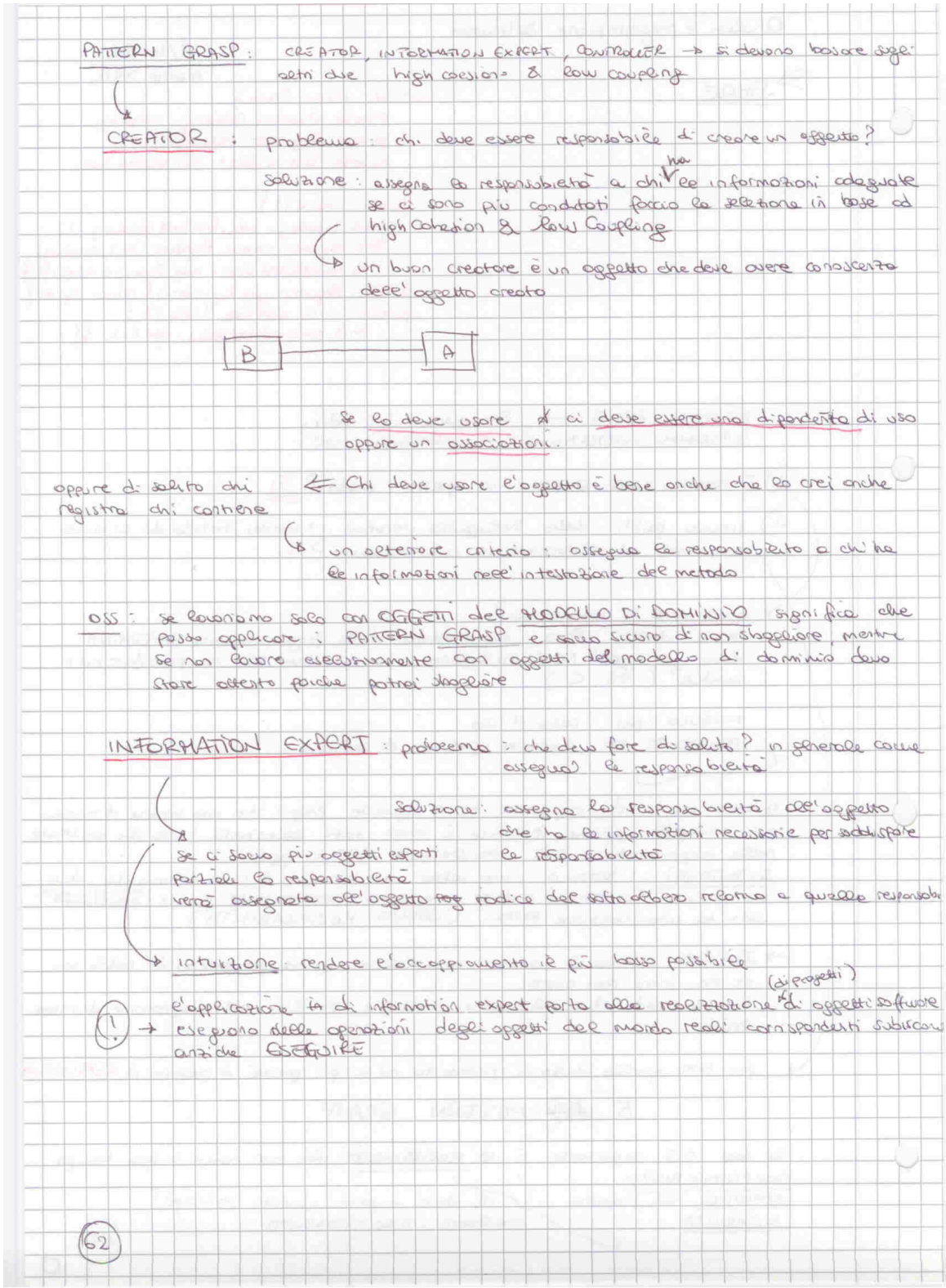
SALESLINEITEM :

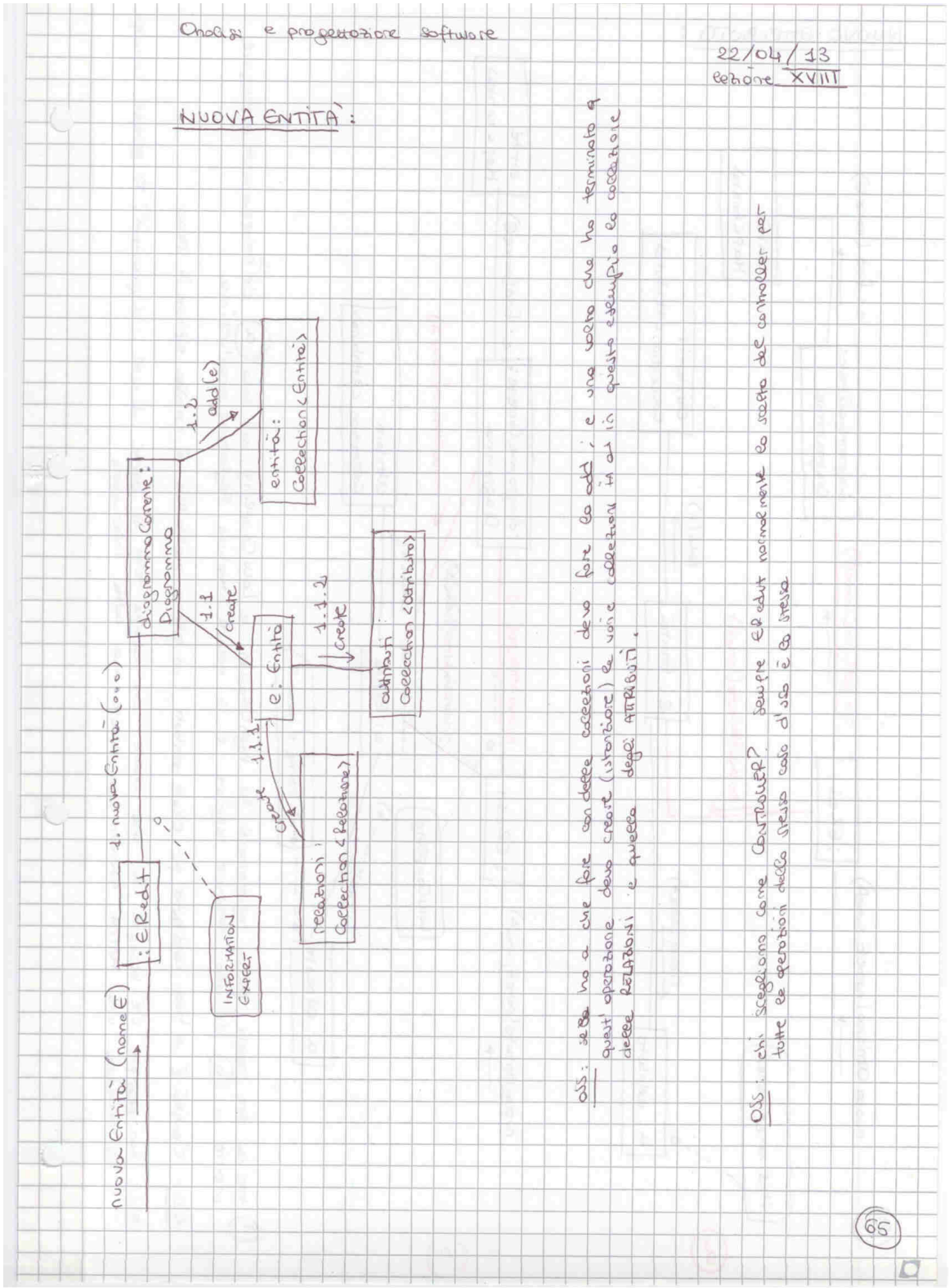
```
public class SalesLineItem {
  private int quantity;
  private ProductDescription description;
  public SalesLineItem(ProductDescription desc, int qty)
    this.description = desc;
    this.quantity = qty; }
  public Money getSubtotal(){
    return description.getPrice().times(quantity)}
```

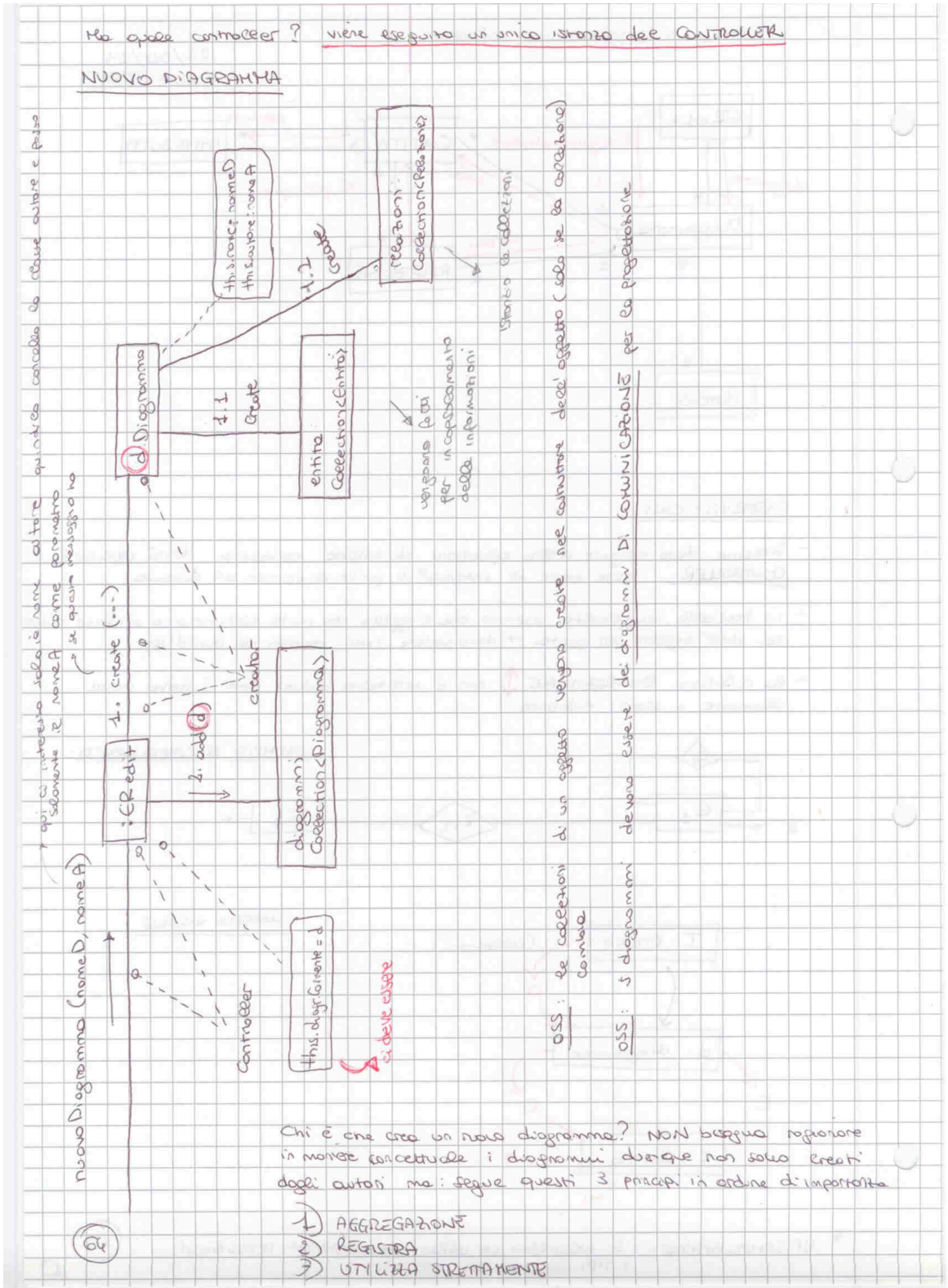
CASHPAYMENT :

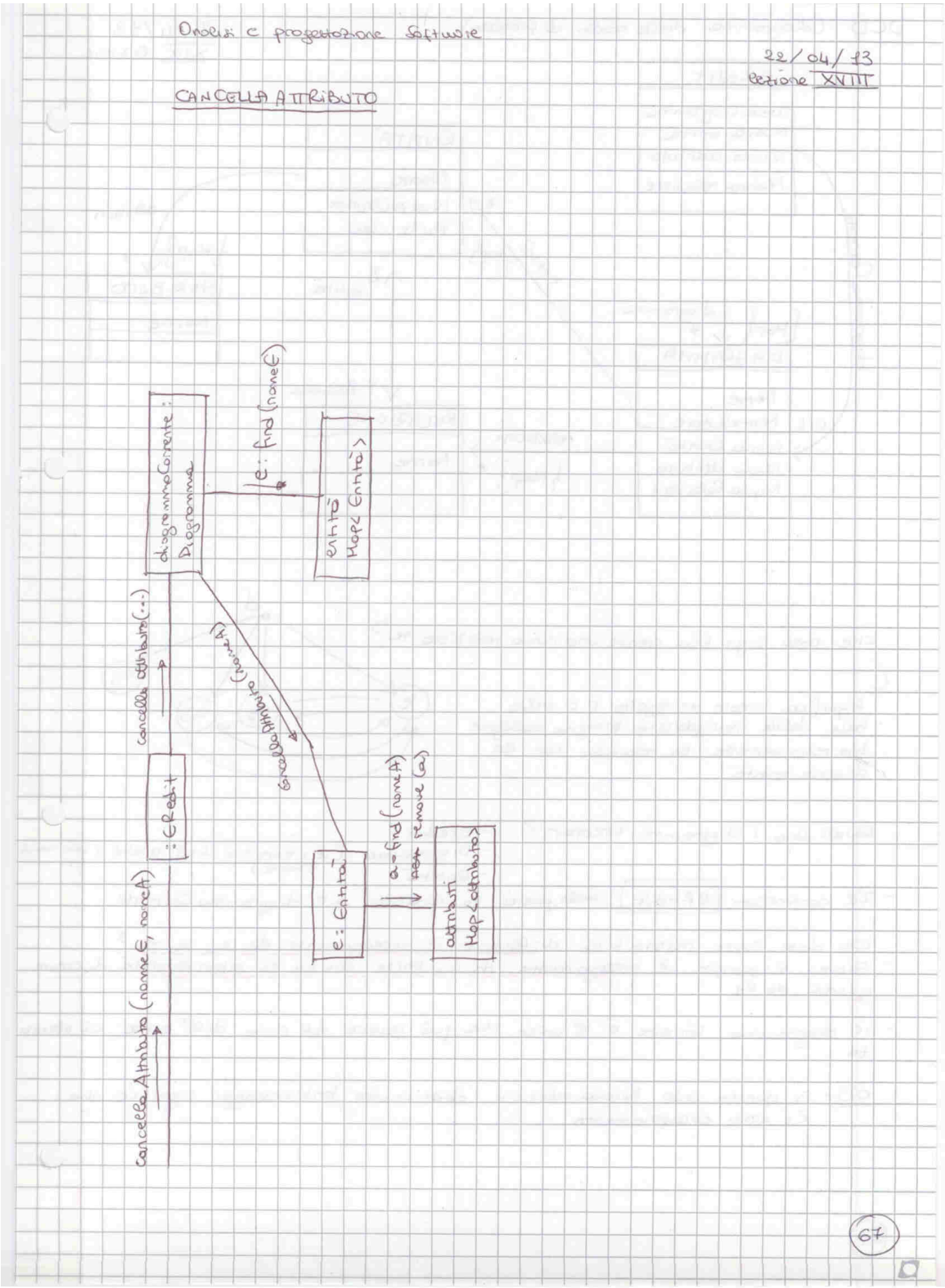
```
public class CashPayment {
  private Money amount;
  public CashPayment(Money cashTendered){
    amount = cashTendered; }
  public Money getAmount(){
    return amount; }}
```

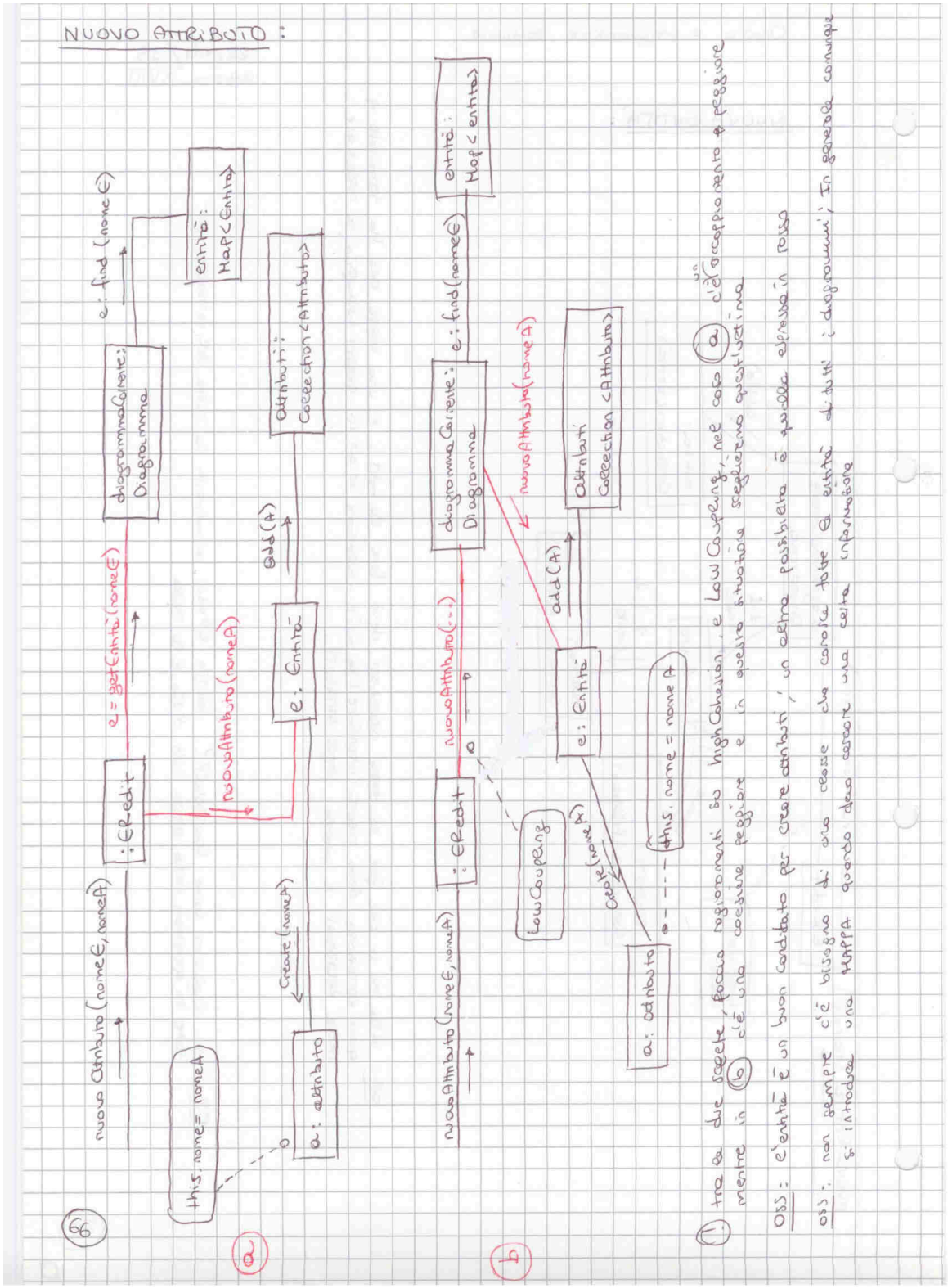

Analisi e progettazione software
22/04/13
Lezione XVIII
ERedit
Diagramma Corrente
ENTITÀ
attributi
ATTRIBUTO
diagramme
entità partecipanti
Diagramma
entità
RELAZIONE
AUTORE
OSSERVAZIONI
- Siccome devo eseguire delle operazioni di sistema certamente dovrò applicare CONTROLLER, deve essere ie sistema® ie punto di accesso al sistema.
- in base alla navigabilità, esprimo che e'oggetto che porta contiene una variabile d'istanza dell' oggetto che punta → devo vedere tutti i metodi dei coni d'uso
- La relazione BIDIREZIONALE ↕ non è veramente bidirez. però è stata messa solamente a scopo didattico
ESEMPIO DI RIFERIMENTO:
A
MODELLO VISUALE
I : Edit
(collezioni)
d : diagramma
e : Entità
* HP. SEMPLIFICATIVE: si IGNORANO LE VARIE SEMPLIFICAZIONI DESCRIZIONI
i TIPI degli attributi etc etc
63

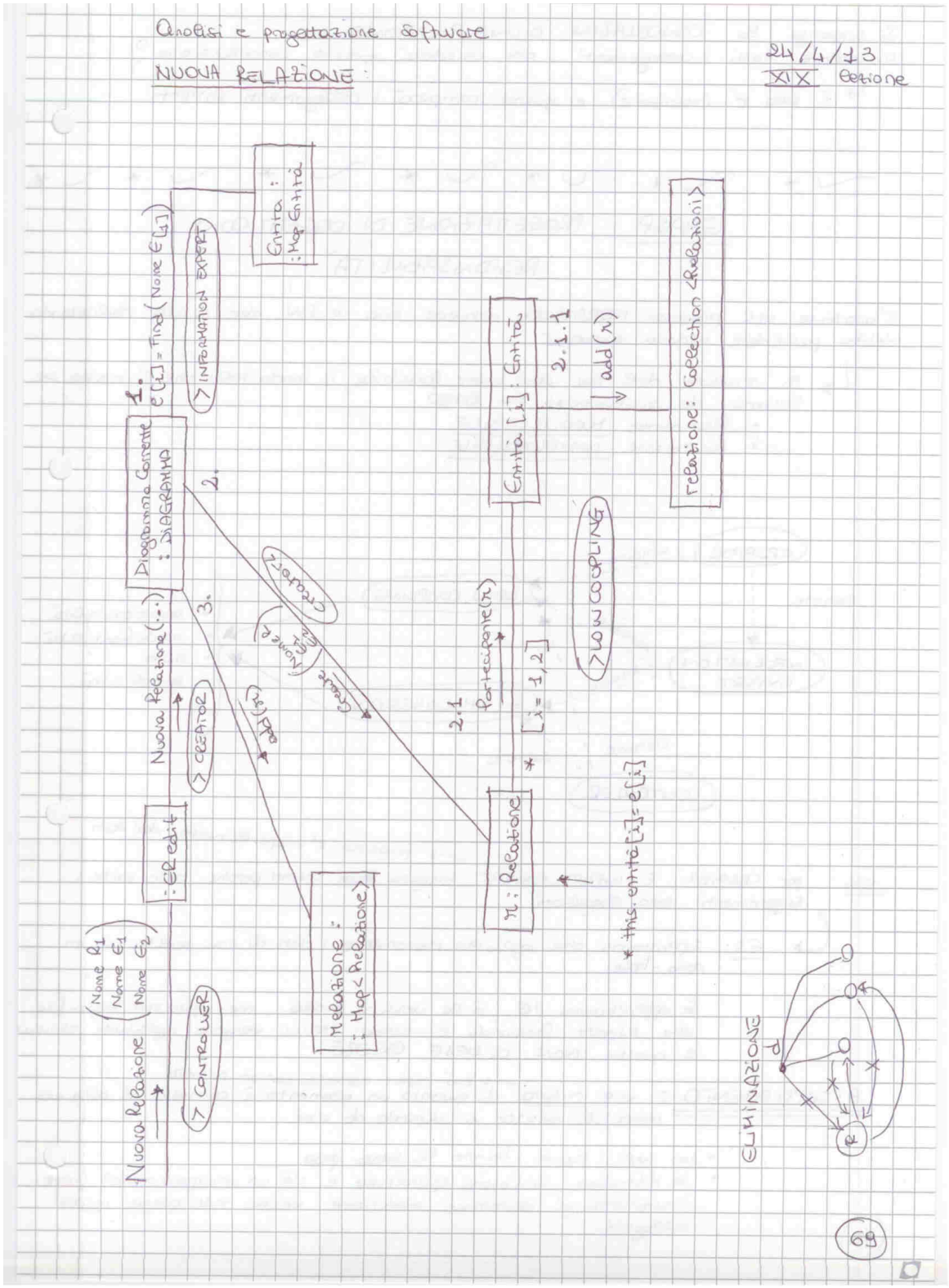

PATTERN GRASP: CREATOR, INFORMATION EXPERT, CONTROLLER → si devono basare oggetti due high cesion & low coupling
CREATOR: problema: chi deve essere responsabile di creare un oggetto?
soluzione: assegna la responsabilità a chi ha le informazioni adeguate se ci sono più candidati faccio la selezione in base ad high Cohesion & low Coupling
un buon creatore è un oggetto che deve avere conoscenza dell' oggetto creato
B
A
se lo deve usare ci deve essere una dipendenza di uso oppure un associazione.
oppure di solito chi registra chi contiene ⇐ Chi deve usare l'oggetto è bene anche che lo crei anche
un ulteriore criterio: assegna la responsabilità a chi ha le informazioni nell' intestazione del metodo
OSS: se lavoriamo solo con OGGETTI del MODELLO DI DOMINIO significa che posso applicare i PATTERN GRASP e sono sicuro di non sbagliare, mentre se non lavoro esclusivamente con oggetti del modello di dominio devo stare attento perchè potrei sbagliare
INFORMATION EXPERT: problema: che devo fare di solito? in generale come assegno la responsabilità
soluzione: assegno la responsabilità dell' oggetto che ha le informazioni necessarie per soddisfare la responsabilità
se ci sono più oggetti esperti parziali la responsabilità verrà assegnata dell' oggetto radice del sotto albero relativo a quelle responsab
intuizione: rendere l'accoppiamento il più basso possibile
(di progetti)
1. → l'applicazione di information expert porta alla realizzazione di oggetti software che eseguono delle operazioni degli oggetti del mondo reale corrispondenti subito anzichè ESEGUIRE
62


Analisi e progettazione software
22/04/13
Lezione XVIII
NUOVA ENTITÀ:
nuova Entità (nomeE)
: ERedit
1. nuovaEntità (...)
diagramma corrente: Diagramma
1.1 create
1.2 add(e)
entità: Collection<Entità>
e: Entità
1.1.1 create
attributi: Collection<Attributo>
1.1.2 create
relazioni: Collection<Relazione>
Information Expert
Oss: se ho a che fare con delle collezioni devo fare da colui che ha terminato a quest'operazione dove creare (istanziare) la vera collezione in cui in questo esempio è contenuta e deve RELAZIONI e questa deve: ATTRIBUTI.
Oss: chi seglama come Controller? sempre ER edit normalmente lo stato del controller per tutte le operazioni dello stesso caso d'uso è la stessa
65


Ma quale controller? viene eseguito un unico istanza del CONTROLLER
NUOVO DIAGRAMMA
Diagramma
entità
Collection<Entità>
Reazioni
Collection<Reazioni>
this.create(nomeD, this.nomeA)
1:1 Create
1:1 Create
Usando la collezioni
vengono gli eventi per la comunicazione della informazioni
nuovoDiagramma (nomeD, nomeA)
1. create (...)
2. add(d)
iER edit
diagrammi
Collection<Diagramma>
creator
controller
this.diagrammi = d
si deve essere
OSS: le collezioni di un oggetto vengono create nel costruttore dell'oggetto (solo se ha collezioni)
OSS: i diagrammi devono essere dei diagrammi di COMUNICAZIONE per la progettazione
Chi è che crea un nuovo diagramma? NON bisogna ragionare in maniera concettuale i diagrammi devono non solo creati dagli autori ma: segue questi 3 principi in ordine di importanza
1) AGGREGAZIONE
2) REGISTRA
3) UTILIZZA STRETTAMENTE

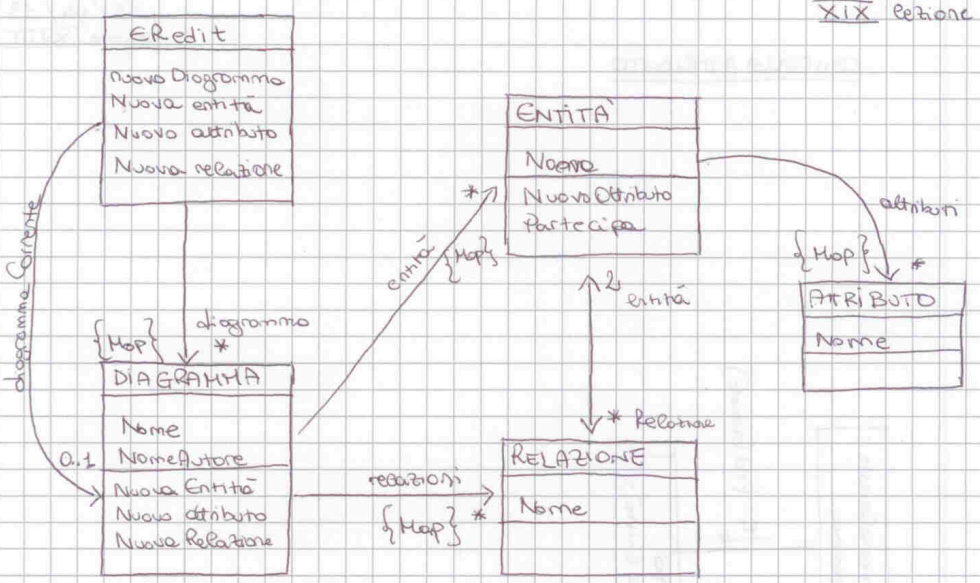

Analisi e progettazione Software
22/04/13
Lezione XVIII
CANCELLA ATTRIBUTO
dragrammaCorrente : Diagramma
e: Find (nomeE)
entità Map<Entità>
cancella Attributo(..,..)
:Edit
cancellaAttributo(nomeA)
e: Entità
a = Find (nomeA)
remove (a)
attributi Map<attributo>
cancella Attributo (nomeE, nomeA)
67


NUOVO ATTRIBUTO :
nuovoAttributo(nomeE, nomeA)
this.nome = nomeA
a: attributo
create(nomeA)
e: Entità
add(A)
attributi: Collection<Attributo>
nuovoAttributo(nomeA)
e = getEntità(nomeE)
e: find(nomeE)
diagramma Corrente: Diagramma
entità: Map<Entità>
: EPedit
a)
b)
66
nuovoAttributo(nomeE, nomeA)
: EPedit
LawCoupling
create(nomeA)
e: Entità
add(A)
attributi: Collection<Attributo>
this.nome = nomeA
a: attributo
1) tra le due soluzioni, focus migliramenti su High Cohesion e Low Coupling, nel caso a) c'è accoppiamento e peggiore mentre in b) c'è una coesione peggiore e la nostra situazione seguiremo questi ultimi
OSS: l'entità è un buon candidato per creare attributi, un altra possibilità è quella alternativa di fare
oss: non sempre c'è bisogno di una classe che controlla tutte le altre, dipendenze comunque si rimane una MAPPA quando deve essere una certa informazione


Analisi e progettazione software
NUOVA RELAZIONE
24/4/13
XIX lezione
Entità : Map<Entità>
> INFORMATION EXPERT
Diagramma Corrente : DIAGRAMMA
e[i] = find(Nome e[i])
1.
2.
3.
Nuova Relazione(...)
> CREATOR
:ER edit
Nuova Relazione
Nome R1
Nome E1
Nome E2
> CONTROLLER
r : Relazione
Creatore
Nome R
Tipo
Nome E1
Nome E2
add(r)
r relazione : Map<Relazione>
2.1
Partecipante(r)
[i = 1,2]
*
π : Relazione
Entità [i] : Entità
2.1.1
add(r)
Relazione: Collection<Relazioni>
> LOW COUPLING
* this.entità[i] = e[i]
ELIMINAZIONE
d
r
69

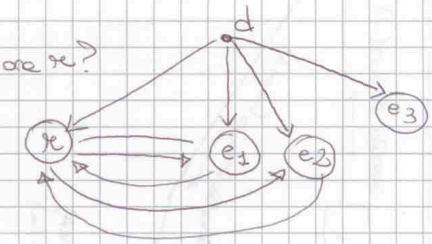
DCD (diagramma delle classi di progetto)

24/4/13
XIX lezione



Che cosa significa creare una nuova relazione re?

Significa creare un oggetto r e nella base delle navigabilità bisogna collegare bidirezionalmente la relazione con le diverse entità



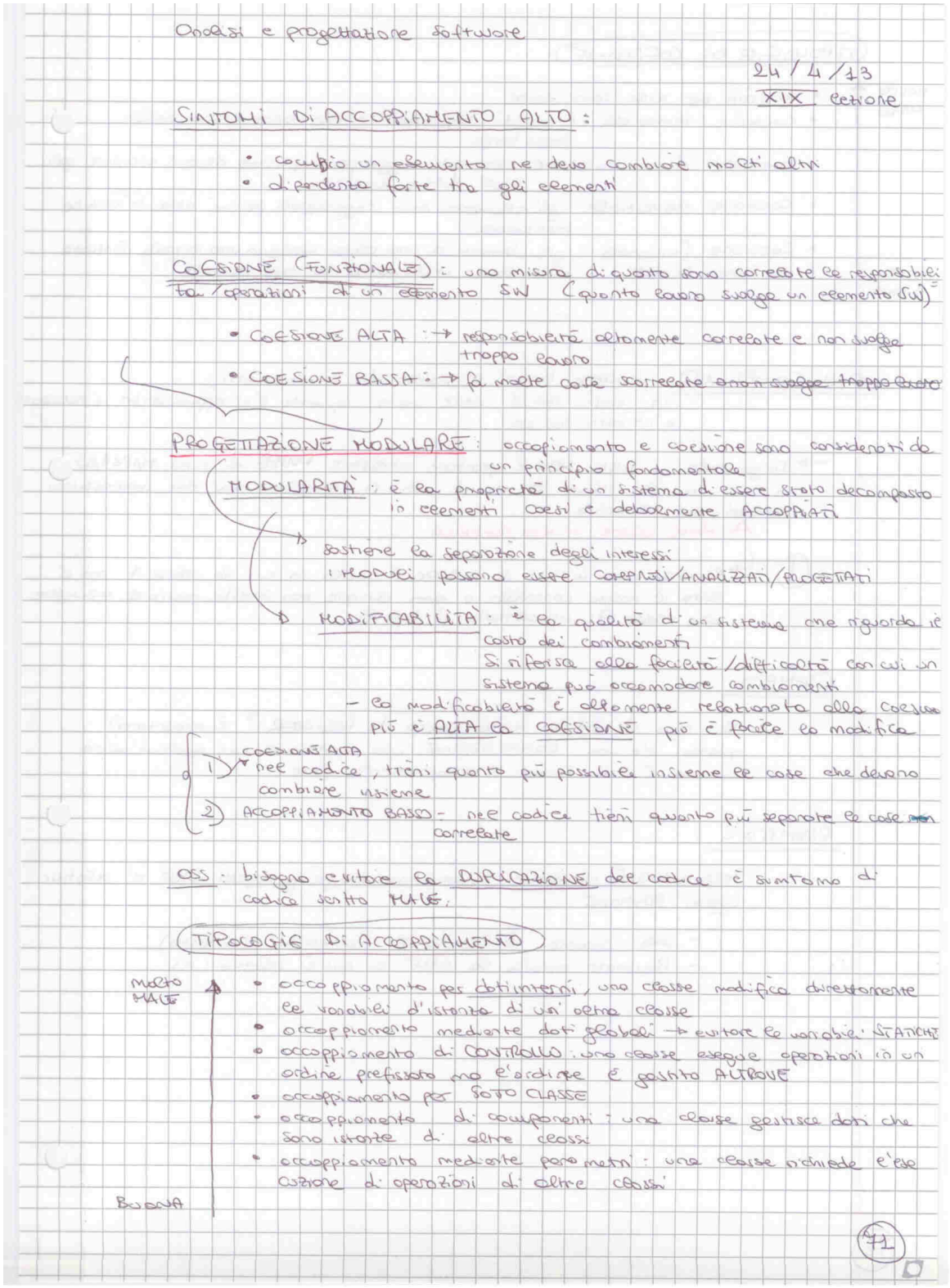
Quali sono i diagrammi creator ? - ERedit
- Diagramma (è creatore di agr. perché è una composizione)
- Entità

- Le controller ERedit delega la creazione a e diagramma Corrente

Chi deve essere responsabile di formare il collegamento da e1 a re ?
Facile è sempre il collegamento da cui parte che ha la responsabilità di creare quindi da e1

- il diagramma corrente è l'unico che può possedere dal nome delle entità ed riferimento.

OSS: in questo caso Nuova relazione deve inviare tanti messaggi significa che c'è stato accoppiamento



Analisi e progettazione software

24/4/13
XIX Lezione

SINTOMI DI ACCOPPIAMENTO ALTO:

- cambio un elemento ne devo cambiare molti altri
- dipendenza forte tra gli elementi

COESIONE (FUNZIONALE): una misura di quanto sono correlate le responsabilità/operazioni di un elemento SW (quanto lavoro svolge un elemento SW)

- COESIONE ALTA: → responsabilità altamente correlate e non svolge troppo lavoro
- COESIONE BASSA: → fa molte cose scorrelate ~~e non svolge troppo lavoro~~

PROGETTAZIONE MODULARE: accoppiamento e coesione sono considerati da un principio fondamentale

MODULARITÀ: è la proprietà di un sistema di essere stato decomposto in elementi coesi e debolmente ACCOPPIATI

→ sostiene la separazione degli interessi
i moduli possono essere CAPITI/ANALIZZATI/PROGETTATI

→ MODIFICABILITÀ: è la qualità di un sistema che riguarda il costo dei cambiamenti
Si riferisce alla facilità/difficoltà con cui un sistema può accomodare cambiamenti
- la modificabilità è altamente relazionata alla coesione più è ALTA la COESIONE più è facile la modifica

1) COESIONE ALTA
nel codice, tieni quanto più possibile insieme le cose che devono cambiare insieme
2) ACCOPPIAMENTO BASSO - nel codice tieni quanto più separate le cose ~~non~~ correlate

OSS: bisogna evitare la DUPLICAZIONE del codice è sintomo di codice scritto MALE.

TIPOLOGIE DI ACCOPPIAMENTO

MOLTO MALE

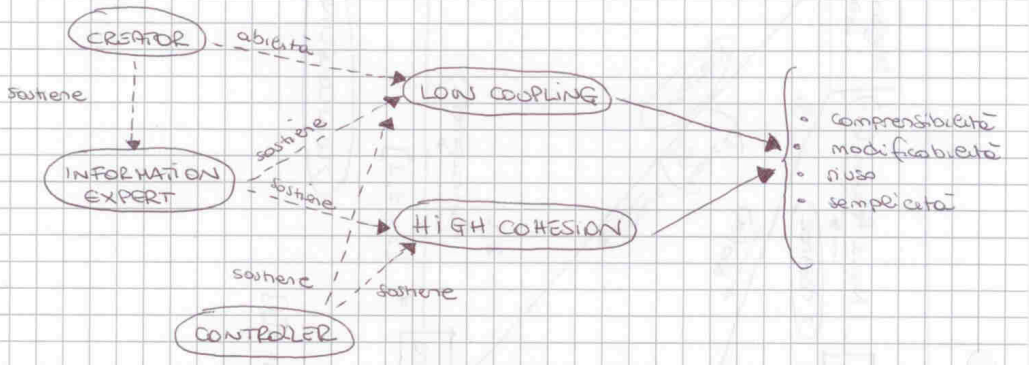
- accoppiamento per dati interni, una classe modifica direttamente le variabili d'istanza di un'altra classe
- accoppiamento mediante dati globali → evitare le variabili STATICHE
- accoppiamento di CONTROLLO: una classe esegue operazioni in un ordine prefissato ma l'ordine è gestito ALTROVE
- accoppiamento per SOTTOCLASSE
- accoppiamento di componenti: una classe gestisce dati che sono istanze di altre classi
- accoppiamento mediante parametri: una classe richiede l'esecuzione di operazioni di altre classi

BUONA

Si supponga la CANCELLAZIONE DI UNA RELAZIONE : cancellare tutti i collegamenti, chi chiederà questa cancellazione?
↳ è solo R (relazione) e quindi romperà i collegamenti entrati

GRASP : PROGETTAZIONE DI OGGETTI CON RESPONSABILITÀ

L'adozione del processo ITERATIVO richiede che il SW che si sta realizzando debba possedere alcune qualità
↳ la struttura del SW deve essere flessibile, in modo tale che l'impatto sul sistema dei cambiamenti sia BASSO
- facilmente MODIFICABILE
- facilmente COMPRENSIBILE



OSS: per CREATOR E INFORM. EXPERT bisogna stare attenti perché certe volte i suggerimenti sono sbagliati → favorisce l'incapsulamento dei dati
↳ ES: supponiamo che vogliamo memorizzare i dati di una vendita in un data-base
Se applichiamo IE è la vendita stessa, ma questo significa che aspetti funzionali e aspetti tecnici vengono implementati insieme e questo NON DEVE ESSERE

ACCOPPIAMENTO : una misura (può essere espresso con un numero) di quanto un elemento è connesso ad altri elementi, li conosce o dipende da essi
- non tutti i legami hanno lo stesso peso
- se dobbiamo cambiare qualcosa e c'è un accoppiamento forte sicuramente ci dovremmo modificare anche tutti quegli oggetti collegati

Analisi e progettazione software

2/5/13
XX lezione

→ Si fa riferimento alla COESIONE della connettività degli oggetti

COESIONE: (funzionale) se un oggetto fa tutte cose simili va bene
- cerco di assegnare le responsabilità in modo tale che i cambiamenti sono organizzati.
Se devo cambiare qualcosa insieme in due oggetti è meglio che questi ultimi siano affini e molto vicini

nella REALTÀ: una persona poco COESA è quella a cui viene chiesto di fare troppo lavoro o se gli viene chiesto di fare abbastanza lavoro in aree diverse

OSS: high Coesion e Low coupling sono difficile da applicare direttamente ecco perché vengono definiti anche IE, ed altri ANTI-PATTERN da evitare.

ANTI-PATTERN: DUPLICAZIONE CODICE

→ lo stesso codice è presente in classi diverse, in diversi metodi ovvero SE HO UNA CLASSE CHE È SIMILE AD UN ALTRA E FACCIO COPIA-INCOLLA E POI MODIFICO, (questo è un male)
- è una parte del codice è una cosa di cui mi occorgo quando la scrivo
- il codice diventa più difficile da capire e l'errore è più difficile da correggere
- inoltre code duplication è una netta violazione del codice di accoppiamento perché codice simile è fortemente accoppiato, e anche di high coesion perché ho mancato di mettere il codice a fattor comune.

non sono riuscito a trovare delle astrazioni che abbassa la coesione

→ si risolve mettendo a fattore comune il codice

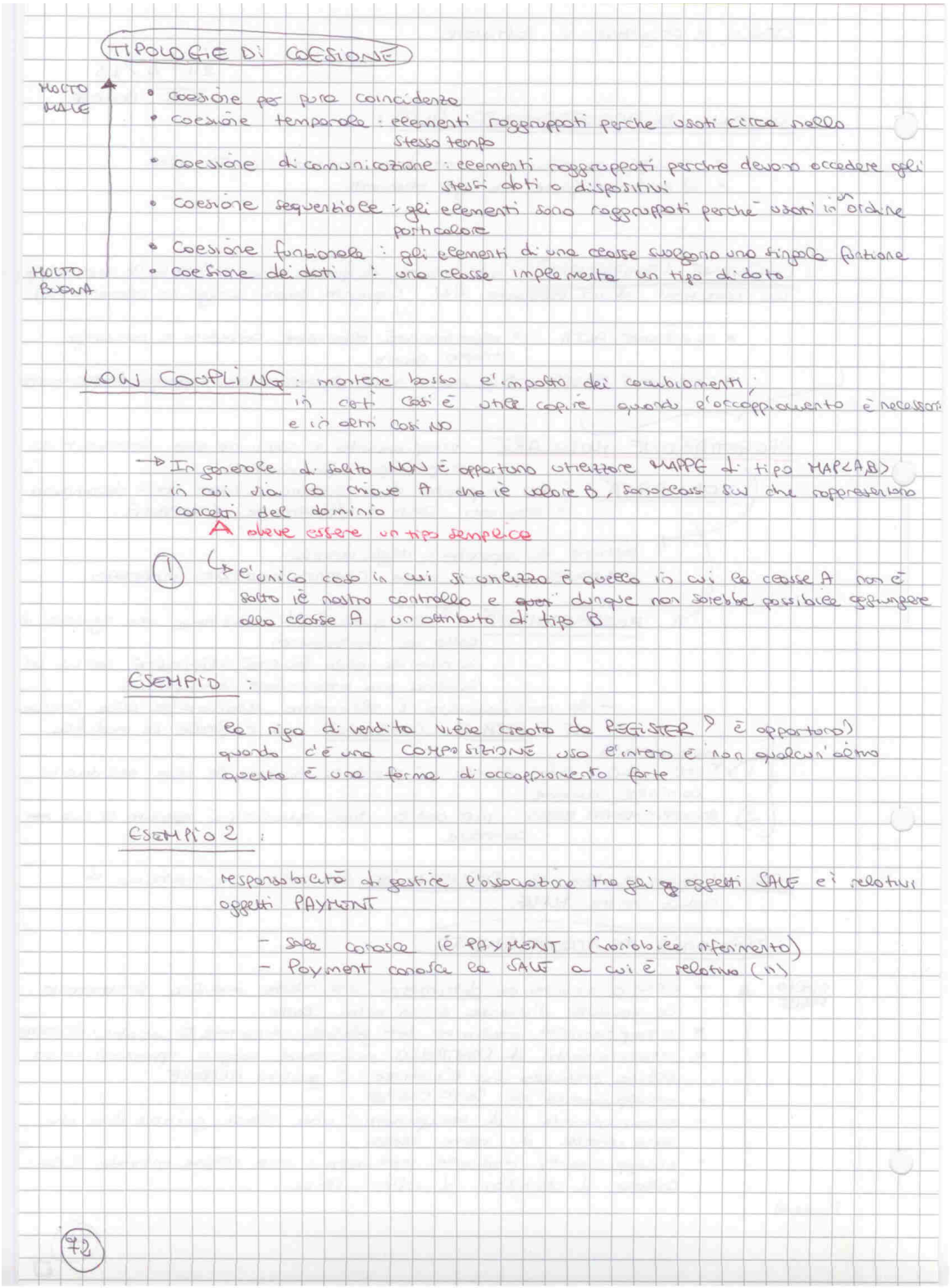
SRP: [SINGLE RESPONSABILITY PRINCIPLE]

è un principio, quindi da solo un suggerimento non una soluzione;
→ UNA CLASSE DOVREBBE AVERE UNA SOLA RAGIONE PER CAMBIARE (una sola responsabilità) → altamente coesa

OSS: se verifico una violazione di SRP sono sicuro che violerò anche high-coesion (è più facile verificare violaz. di SRP che HC)

ESEMPIO: A deve conoscere un oggetto di tipo B: due modi per farlo
1) o A memorizza un riferimento a B
2) o A memorizza un valore che identifica un oggetto di tipo B ovvero una chiave esterna (molto per la modellazione di domino)
→ In questo caso se cambio a dovrò cambiare anche il secondo

43



TIPOLOGIE DI COESIONE

MOLTO MALE ↑ MOLTO BUONA

- coesione per puro coincidenza
- coesione temporale: elementi raggruppati perche usati circa nello stesso tempo
- coesione di comunicazione: elementi raggruppati perche devono accedere agli stessi dati o dispositivi
- coesione sequenziale: gli elementi sono raggruppati perché usati in un ordine particolare
- coesione funzionale: gli elementi di una classe svolgono una singola funzione
- coesione dei dati: una classe implementa un tipo di dato

LOW COUPLING: mantiene basso l'impatto dei cambiamenti, in casi casi è utile capire quando l'accoppiamento è necessario e in altri casi NO

→ In generale di solito NON è opportuno utilizzare MAPPE di tipo MAP<A,B) in cui via la chiave A che è valore B, sarebbero sui che rappresentano concetti del dominio

A deve essere un tipo semplice

(!) ↳ è unico caso in cui si eccezza è quello in cui la classe A non è sotto il nostro controllo e quindi non sarebbe possibile aggiungere alla classe A un attributo di tipo B

ESEMPIO:

La riga di vendita viene creata da REGISTER ? è opportuno) quando c'è una COMPOSIZIONE uso e intero e non qualcun'altro questa è una forma di accoppiamento forte

ESEMPIO 2:

responsabilità di gestire l'associazione tra gli oggetti SALE e i relativi oggetti PAYMENT

- Sale conosce il PAYMENT (navigabile afferrato)
- Payment conosce la SALE a cui è relativo (1)

Analisi e progettazione software

3/5/13
XX lezione

un controller è il primo oggetto oltre lo strato UI che riceve e coordina un'operazione di sistema
- la sua interfaccia è costituita da operazioni di sistema
- queste operazioni di sistema sono chiamate da oggetti dello strato UI.

OSS: IL CONTROLLER deve sempre delegare le informazioni (le operazioni) le informazioni TRANSIENTI vengono gestite dal controller

(!) è importante che ci sia un controller per ogni attore primario del sistema che implementa un interfaccia ⇒ è la soluzione MIGLIORE ma noi UTILIZZEREMO NEL CORSO SOLO FACADE CONTROLLER

SERVIZIO STATELESS: ciascuna richiesta non dipende dalle altre e quindi non mi interessa quale client ha richiesto, perché il sistema risponderà sempre allo stesso modo
ESEMPIO: "dimmi che ore sono"

SERVIZIO STATEFULL: ciascuna richiesta deve tener conto dello stato della sessione (e quindi dipende dalle altre)
ESEMPIO: carrello della spesa su AMAZON, una volta che ho aggiunto gli oggetti che voglio comprare e procedo con la richiesta di pagamento. La richiesta deve sapere qual è lo stato della sessione.

alcuni servizi statefull possono essere implementati con stateless

nel corso noi terremo conto solo di servizi STATEFULL implementati con STATEFULL
Il controller gestisce le informazioni sulla sessione
→ dell'esame no oggetto Sessione metterò direttamente nel controller

SVILUPPO GUIDATO DAI TEST E REFACTORING

"Questo metodo di procedere sostiene il cambiamento?"

Sviluppo guidato dai test è una cosa metodologica e differisce quindi dallo sviluppo con i test.

a questa domanda rispondiamo:

ci sono 2 metodi che abbracciano questo cambiamento

- l'uso dei test automatici
- refactoring

OSS: noi faremo in questo questo corso di ROO, e quindi sviluppo con i test

TEST-DRIVEN DEVELOPMENT: ① ripetutamente il test è scritto prima del codice da testare. ② Immaginando che il codice da testare sia stato già scritto ③ poi si scrive il codice per far passare il test

ESEMPIO: gestire la vendita nel sistema pos

↳ Potrebbe essere opportuno che la vendita sia creata dalla classe CASHIER?
Cassiere ha a che fare con le informazioni del cassiere;

nel momento in cui la specifica cambia nel senso che il cliente arriva paga i suoi articoli con la carta e se ne va
Quindi la classe cassiere deve essere cambiata anche in un altro caso d'uso in cui quest'ultima nemeno compare.
Questa è una CHIARA violazione di SRP

ANTI-PATTERN: BLOB

nel DCD ci sono tante classi molto piccole, però c'è un altra classe che è poco COESA quindi molto grande. Quest'oggetto qui è un blob
→ Non bisogna pensare che la coesione è una media delle singole coesioni
↳ da COESIONE BASSA e ACCOPPIAMENTO ALTO
OSS: se nella progettazione un diagramma di comunicazione assume una FORMA a "stella" - in cui c'è un elemento centrale che collabora con molti oggetti, chiedendo a ciascuno di fare delle cose piuttosto semplici

CONTROLLER GRASP: lo chiamiamo così perché la parola controller è sovraccarica, ovvero possiede diversi significati, simili fra loro
↳ l'obbiettivo è quello di fare un controller per accoppiamento e coesione (non per controllare le operazioni di sistema)

definisci una INTERFACCIA DI SISTEMA in corrispondenza a ciascun caso d'uso quindi definisci il controller in base ad una delle seguenti scelte.

- [USE-CASE CONTROLLER]
  un controller diverso per ciascuna interfaccia di sistema che la implem.
- [FACADE CONTROLLER]
  - per tutte le interfacce di sistema - che le implementa tutte
  - un controller diverso per ciascun attore primario un FACADE per ciascun attore

SOLUZIONE: assegna la responsabilità sulla base di due criteri

OSS: utilizzeremo sempre un FACADE controller: rappresenta il sistema intero oppure un attore

1) APP. STAND-ALONE: (gestisce i dati di un singolo utente) in questo caso il CONTROLLER è l'oggetto che rappresenta l'intero sistema - applicazione
2) APP. CLIENT-SERVER: (più utenti possono usare il sistema in modo concorrente) in questo CASO il CONTROLLER è un oggetto che rappresenta un p.to di accesso al sistema per uno dei suoi CLIENT

OSS: controller dice chi è il primo oggetto oltre gli strati di dominio

74 LA PRESENTAZIONE dice che quando l'utente fa click ci chiede copiami quali sono i parametri dell'operazione e chiedi al controller di eseguire quell'operazione al controller

Analisi e progettazione del software
Anno Accademico 2012-2013
Homework 3 – Progettazione

Regole: **SCRIVERE IN MODO LEGGIBILE E NON AMBIGUO**. Scrivere il proprio nome su ciascun foglio utilizzato, in alto a destra. Accanto allo svolgimento di ciascun esercizio o domanda va indicato chiaramente l'esercizio o la domanda a cui lo svolgimento è relativo (ad esempio, Esercizio C1, diagramma delle classi di progetto, oppure Esercizio C2, domanda C2.1). Alle domande in cui è prevista una risposta binaria SI/NO ("è opportuno fare questo?") è necessario scrivere in modo esplicito prima la risposta alla domanda (SI oppure NO) e poi dare una breve motivazione della risposta.

In questo homework si faccia ancora riferimento ai requisiti per il sistema **AcmeArti**, descritti in un documento separato e già utilizzati per gli homework precedenti.

Ipotesi di lavoro, valide per tutti gli esercizi di progettazione.

- In tutti gli esercizi che seguono, si faccia l'ipotesi che il sistema in discussione gestisca i propri dati solo in memoria principale. Si supponga anche che durante il caso d'uso di avviamento vengano creati e caricati in memoria tutti gli oggetti le cui informazioni siano già effettivamente disponibili al momento dell'avviamento.
- Per ciascuna operazione di sistema va creato un diagramma di interazione che descrive l'interazione relativa alla trasformazione (cambiamento di stato) provocata dall'operazione di sistema. Per quanto riguarda invece le relative risposte (interrogazioni) eventualmente restituite dal sistema, se non è richiesto esplicitamente allora non bisogna mostrare nei diagrammi di interazione né il calcolo dei dati da restituire né la loro visualizzazione. Tuttavia, per le risposte del sistema, è comunque necessario verificare che i dati da restituire possano essere (facilmente) calcolati sulla base delle navigabilità tra gli oggetti che sono state progettate (vedi anche il punto successivo).
- **Le soluzioni individuate dovranno essere compatibili (in particolare in termini di visibilità, ovvero di navigabilità delle associazioni) con le realizzazioni di tutti i casi d'uso mostrati (UC1-UC4).**
- **Nei diagrammi di interazione, mostrare IN MODO ESPLICITO: tutti i MESSAGGI scambiati tra oggetti, tutte le CREAZIONI di oggetti e tutte le FORMAZIONI e ROTTURE di collegamenti.**
- Nei diagrammi di interazione, motivare le scelte di progetto fatte indicando i pattern GRASP e GoF applicati.
- Nei diagrammi delle classi di progetto, mostrare: (1) per ciascuna classe: il nome della classe, i nomi dei suoi attributi, i nomi delle sue operazioni; e (2) per ciascuna associazione e ciascuna sua estremità navigabile: la freccia di navigabilità, il nome dell'estremità, la molteplicità e, in caso di associazione navigabile a molti, il tipo di collezione scelta.

Esercizio C1 (Progettazione)
Fare la progettazione a oggetti per il sistema in discussione, relativamente al caso d'uso UC1 (*Inserimento edizioni corsi*), come segue:

- Mostrare i diagrammi di interazione relativi a tutte le operazioni di sistema per il caso d'uso UC1.
- Mostrare il corrispondente diagramma delle classi di progetto.

Esercizio C2 (Domande)
Relativamente al precedente esercizio C1, rispondere alle seguenti domande (rispondere in modo sintetico, scrivendo circa 3-5 righe per ciascuna domanda):

- (C2.1) In quale operazione di sistema è più opportuno creare nuovi oggetti di tipo "Docente"?
- (C2.2) In quale operazione di sistema è più opportuno creare nuovi oggetti di tipo "Edizione corso"?
- (C2.3) E' opportuno che gli oggetti "Edizione corso" siano creati da oggetti "Docente"? Motivare la risposta.
- (C2.4) Quali sono secondo Creator tutti i possibili candidati creatori di una "Edizione corso"? E perché?
- (C2.5) Quale navigabilità è necessaria per l'associazione "E' competente per" tra le classi "Docente" e "Corso"? Motivare la risposta, anche facendo riferimento a tutti casi d'uso interessati a tale associazione.
- (C2.6) Quale navigabilità è necessaria per l'associazione "E' relativa a" tra le classi "Edizione corso" e "Corso"? Motivare la risposta, anche facendo riferimento a tutti casi d'uso interessati a tale associazione.

Esercizio C3 (Programmazione)
Con riferimento allo svolgimento del precedente esercizio C1, sia *inserisciEdizioneCorso(mese, descrizione)* l'operazione di sistema motivata dal passo 5 del caso d'uso UC1. Scrivere tutto il codice (porzioni di classi, comprendenti solo i metodi d'istanza e le variabili d'istanza necessarie) relativo alla gestione completa dalla sola operazione di sistema *inserisciEdizioneCorso*, in tutte le classi coinvolte.

SVILUPPO GUIDATO DAI TEST

- non scrivere codice di produzione fino a quando non hai scritto un test unitario che fallisce
- non scrivere più di un test unitario di quanto è sufficiente a far fallire la compilazione o l'esecuzione dei test
- non scrivere più codice di produzione di quanto è sufficiente a far passare il test che fallisce

(!) regole semplici ma difficili da applicare

REFACTORING: i test sono un'attività di supporto al refactoring

↳ è un metodo strutturato e disciplinato per riscrivere o ristrutturare del codice esistente senza modificare il suo comportamento esterno applicando piccoli passi di trasformazione, in combinazione con la ripetizione dei test dopo ciascun passo.

- riorganizza il codice per:
  - rimuovere il codice duplicato
  - per aumentare la chiarezza
  - per avere metodi corti

OSS: il refactoring procede per piccoli passi successivi

REFACTORING IDE → (eclipse)

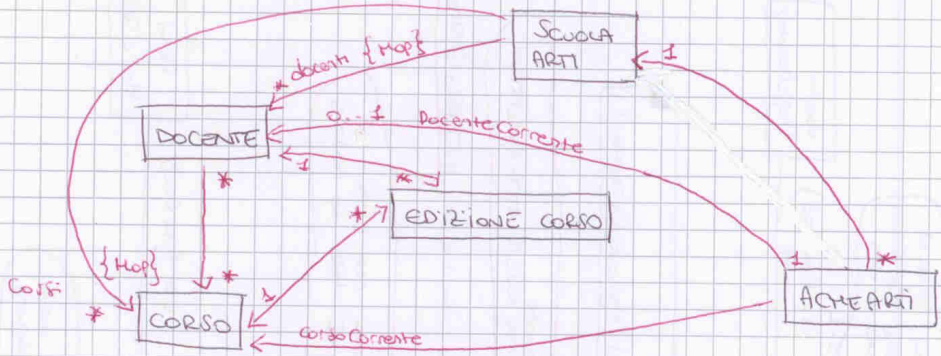
Refactor → extract-method → do il nome → mi crea un nuovo metodo
" - class → " → classe

OSS: i metodi sono più corti e la COESIONE aumenta con gli opportuni riferimenti
e il codice diventa più leggibile

Analisi e progettazione software

6/5/13
Lezione XXI

DCD: HOMEWORK 3



oss: in questo caso l'attore primario è il docente, come lo era anche il cassiere nel sistema POS, solo che quest'ultimo non figurava nella mia progettazione poiché NON DOVEVANO GESTIRE INFORMAZIONI relative a quest'ultimo, mentre con il DOCENTE è così

OPERAZIONI

loop
- loginDocente
- inizioInserimentoEE
- SelezionaCorso
- inserimentoEdizioneCorso
- fineInserimentoEC

oss: per dello rappresentazionale basso molte dei ~~gli~~ diagrammi vengono rappresentati

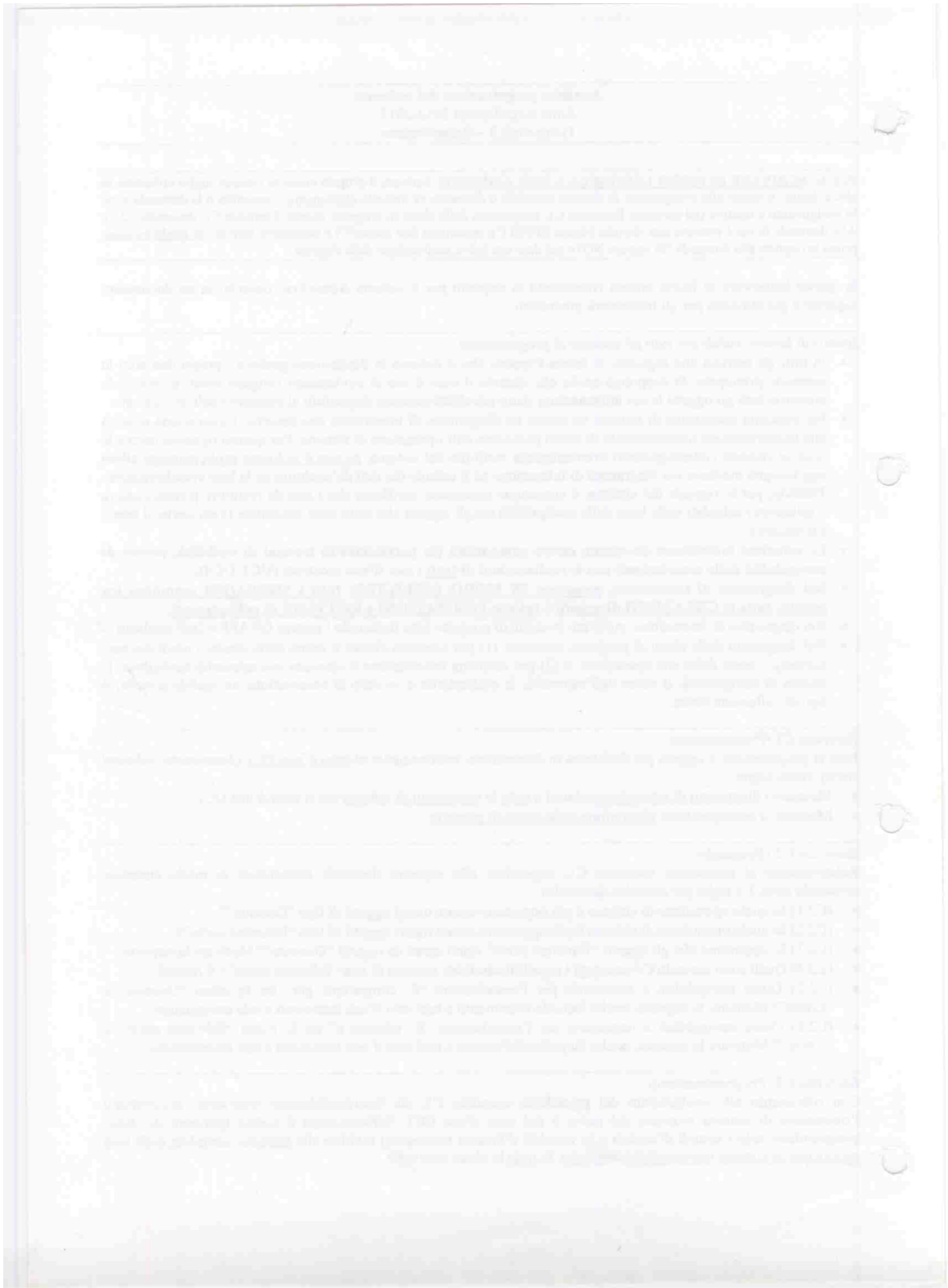
(!) quando si dice che il sistema mostra è tutto il sistema, tutto il software, quindi questo non significa che l'informazione debba essere gestita da oggetti o da una SINGLETON

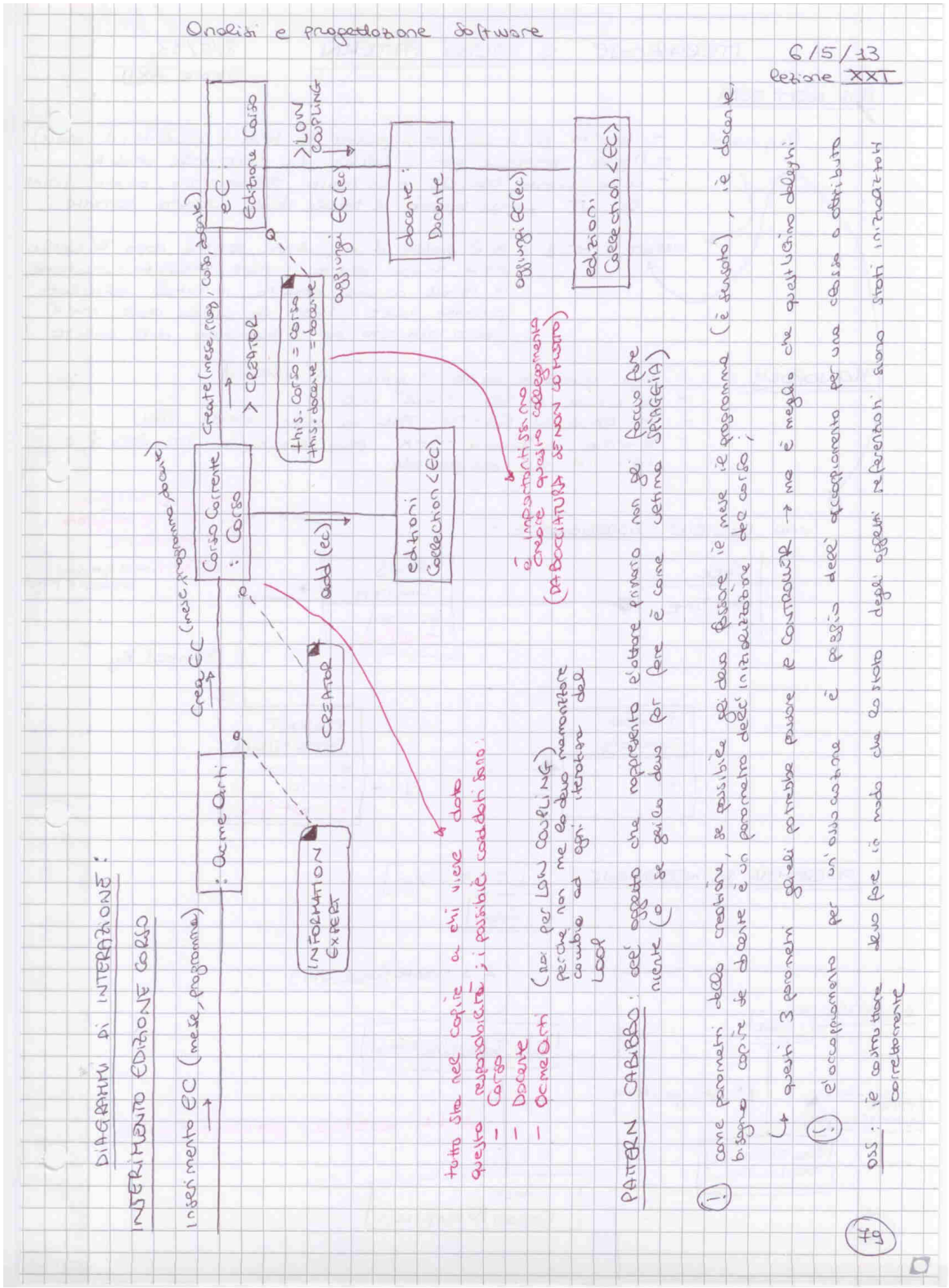
(!) per la navigabilità devo andarmi a leggere i casi d'uso (TUTTI) e vedere se il sistema o qualche oggetto deve avere la VISIBILITÀ adeguata

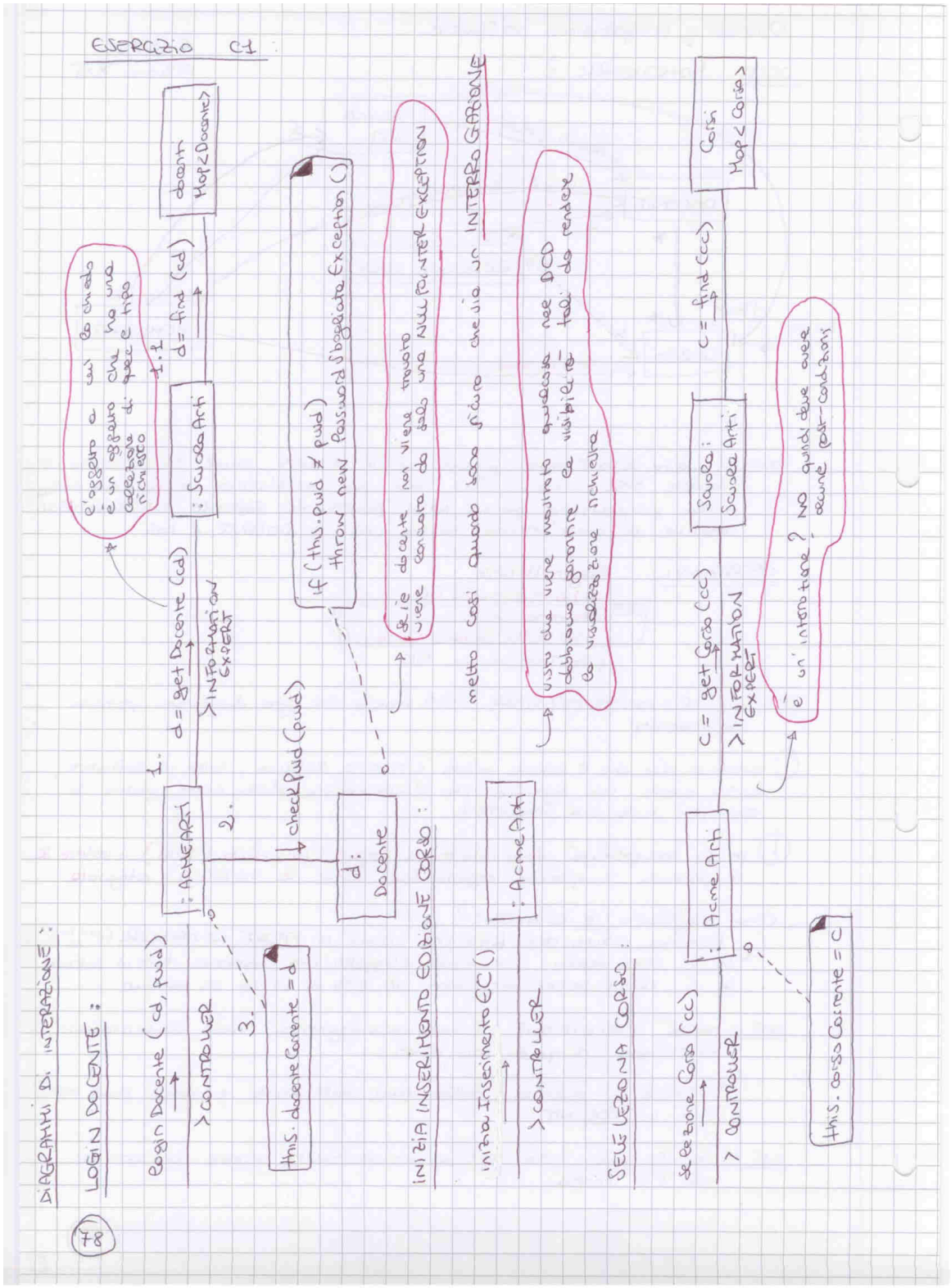
Come scegliamo il controller?
- Partiamo subito dal fatto che usiamo un FACADE CONTROLLER (per tutti i casi d'uso)
- se è stand alone utilizziamo l'oggetto che rappresenta tutto il sistema
- se è client-server scegliamo il p.to di accesso al sistema

oss: nella VALIDAZIONE (nel caso negativo) viene scatenata un'estensione di quel caso d'uso
- Chi è in possesso della fluid dell'utente e della fluid verso? è il DOCENTE

oss: se uso una lista c'è un accoppiamento minore che non se uso una mappa



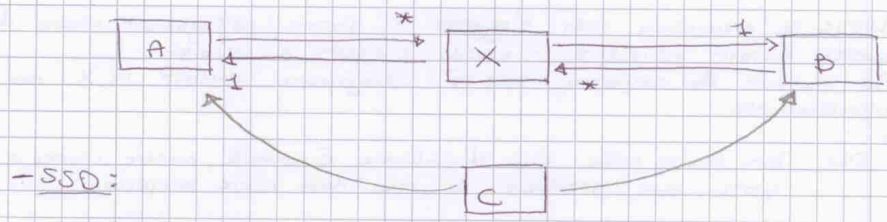

Analisi e progettazione software
6/5/13
Lezione XXI
DIAGRAMMI DI INTERAZIONE:
INSERIMENTO EDIZIONE CORSO
Inserimento EC (mese, programma)
: OmeArti
Crea EC (mese, programma docente)
Corso Corrente : Corso
create(mese, prog, corso, docente)
> CREATOR
EC : Edizione Corso
>LOW COUPLING
aggiungi EC(ec)
docente : Docente
aggiungi EC(ec)
this.corso = corso
this.docente = docente
add (ec)
edizioni : Collection<EC>
edizioni : Collection<EC>
CREATOR
INFORMATION EXPERT
tutto ciò che può capire di chi viene detto questo responsabile: i possibili candidati sono:
- Corso
- Docente
- OmeArti (non per LOW COUPLING perché non ne ho due mantenere coerenza ad ogni iterazione del loop)
è l'implementazione di questo collegamento (DA OCCULTARE SE NON SI USA)
PATTERN CABIBBO: ogni oggetto che rappresenta l'intero primario non può (focus fare niente (o se può fare due (per è come vedrà SPAGHETTI)
(!) come parametri della creazione, se possibile più che due (posizione ie mese ie programma (è sbagliato), ie docente, bisogna capire se docente è un parametro della interazione che cerco;
↳ quindi 3 parametri già è troppo perché posso ricevere → ma è meglio che gestisco delegni
(!) c'è accoppiamento per un'altra cosa è peggio accoppiamento per una classe a attributo
oss: è costituzione uno (ne ho modo che ho stato degli oggetti ne (eventi sono stati inizializzati correttamente


ESERCIZIO C1
DIAGRAMMI DI INTERAZIONE:
LOGIN DOCENTE:
(71/78)
login Docente (cd, pwd)
: AcmeArch
> CONTROLLER
1.
d = getDocente (cd)
> INFORMATION EXPERT
: SuccessArch
d = find (cd)
docenti Map<Docente>
2.
checkPwd (pwd)
d: Docente
3.
this.docenteCorrente = d
e' oggetto a cui ho chiesto e in oggetto che ha una collezione di tipo Docente
1.1
if (this.pwd ≠ pwd)
throw new PasswordDiSbagliataException()
se il docente non viene trovato viene lanciata la NullPointerException
INSERIMENTO CODARSO CORSO:
InizioInserimentoEC()
: AcmeArch
> CONTROLLER
metto così quando sono già dentro un INTERO GRANDE
visto che viene mostrato qualcosa nel DCD dobbiamo garantire la visibilità dei dettagli da rendere
SEGUE LEZIONE CORSO:
setCodiceCorso (cc)
: AcmeArch
> CONTROLLER
c = getCorso (cc)
> INFORMATION EXPERT
Scuola: SuccessArch
c = find (ccc)
Corsi Map<Corso>
è un'interazione? NO quindi deve avere alcune post-condizioni
this.corsoCorrente = c

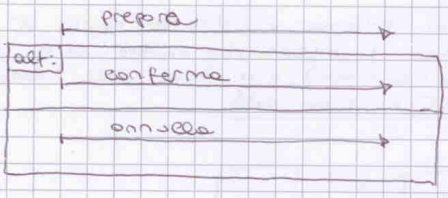
Analisi e progettazione software

9/5/13
Lezione XXIII

ESERCITAZIONE:



- SSD:



Immaginiamo che alla fine del caso d'uso se viene confermato va creato un nuovo oggetto X e collegato ad un oggetto di tipo A e ad un oggetto di tipo B, e dopo vanno collegati da A ad X e da B ad X.

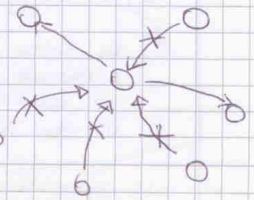
Ci sono tanti modi di procedere:

1) ALTERNATIVA FIGA: durante la preparazione io mi occupo solo di selezionare l'oggetto A e B ma non di creare l'oggetto X (ed anche i rispettivi collegamenti): (accesg. + oggetti)
La creazione viene ritardata, e viene dunque effettuata durante la fase di conferma.

2) ALTERNATIVA ANTICIPATRICE: durante la fase di preparazione viene creato l'oggetto X con ~~tutti~~ i rispettivi collegamenti

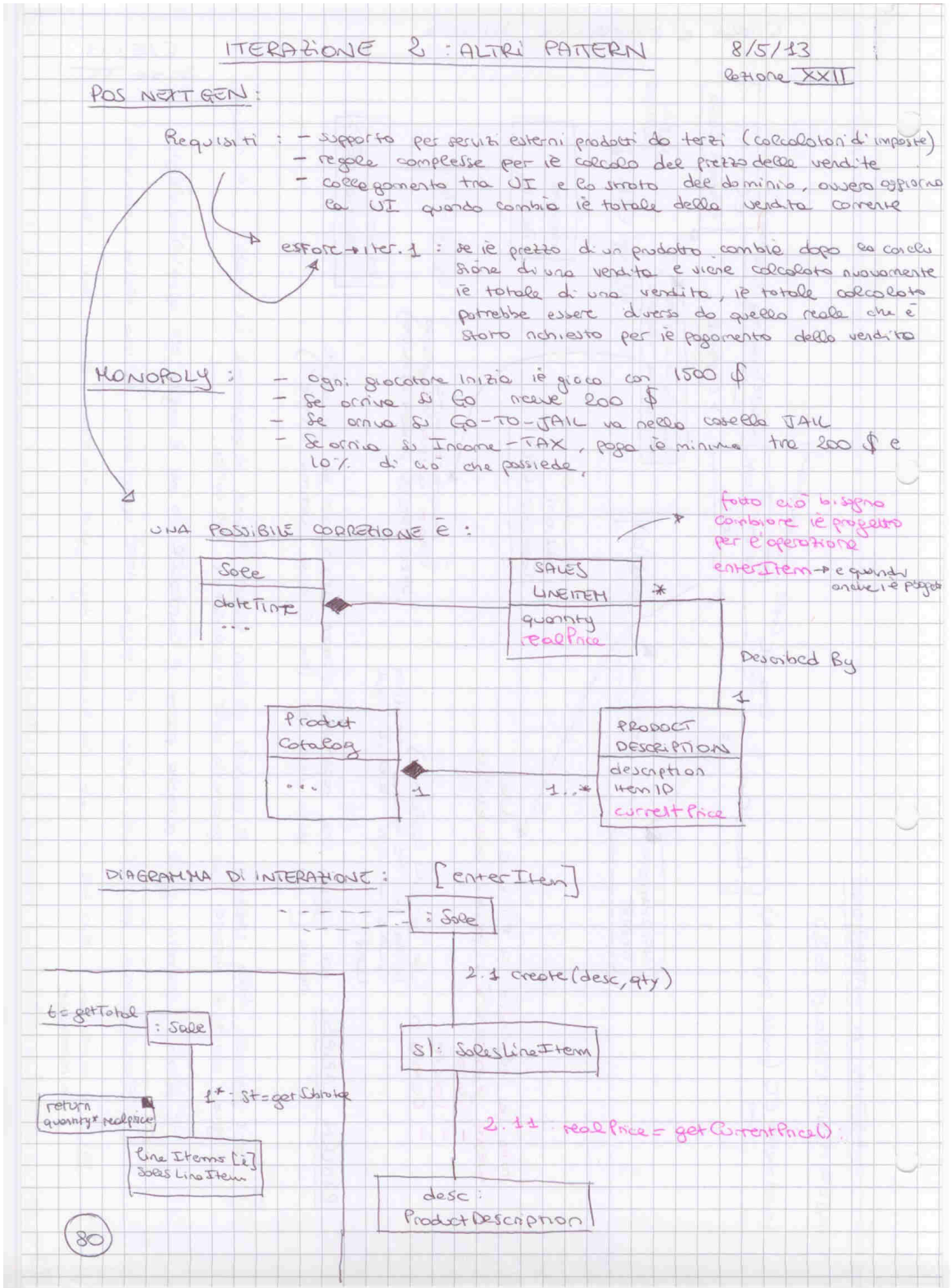
EFFETTO:
- per fare la conferma non devo fare nulla ma per l'annullamento devo rompere tutti i collegamenti ed eliminare l'oggetto
(1) per eliminare l'oggetto X devo solo rompere i collegamenti entranti in X da A e B, perché se non ci sono più i riferimenti in memoria dell'oggetto X, ie GARBAGE eliminerà X che tutti i suoi collegamenti uscenti

EFFETTO:
per annullare non devo fare nulla



ie CONTROLLER si deve ricordare (nel momento della conferma devo fare la creazione) gli oggetti di tipo A e B e probabilmente in questo caso ie controller si dovrà ricordare gli oggetti A e B correnti.

↳ questo è problematico? NO (anche se il controller dovrebbe delegare)


ITERAZIONE 2 : ALTRI PATTERN
8/5/13
Lezione XXII
POS NEXT GEN :
Requisiti : - supporto per servizi esterni prodotti da terzi (calcolatori di imposte)
- regole complesse per il calcolo del prezzo delle vendite
- coordinamento tra UI e lo strato del dominio, ovvero aggiorno la UI quando cambia il totale della vendita corrente
ESFOR → Iter. 1 : se il prezzo di un prodotto cambia dopo la conclusione di una vendita e viene calcolato nuovamente il totale di una vendita, il totale calcolato potrebbe essere diverso da quello reale che è stato richiesto per il pagamento della vendita
MONOPOLY :
- ogni giocatore inizia il gioco con 1500 $
- se arriva su Go riceve 200 $
- se arriva su Go-To-JAIL va nella casella JAIL
- se arriva su Income-TAX, paga il minimo tra 200 $ e 10% di ciò che possiede.
UNA POSSIBILE SOLUZIONE È :
foto ciò bisogno cambiare il progetto per l'operazione enterItem → e quando arrivi è pagato
Sale
dateTime
...
SALES LINEITEM
quantity
realPrice
*
Described By
1
Product Catalog
...
1
1..*
PRODUCT DESCRIPTION
description
itemID
currentPrice
DIAGRAMMA DI INTERAZIONE :
[enterItem]
: Sale
2. 1 create(desc, qty)
t = getTotal
: Sale
1* . st = getSubtotal
return quantity * realPrice
lineItems [i]
SalesLineItem
sl : SalesLineItem
2. 1.1 : realPrice = getCurrentPrice()
desc :
ProductDescription
80

Analisi e progettazione software

9/5/13
Lezione XXIII

RAFFINAMENTO DEL MODELLO DI DOMINIO

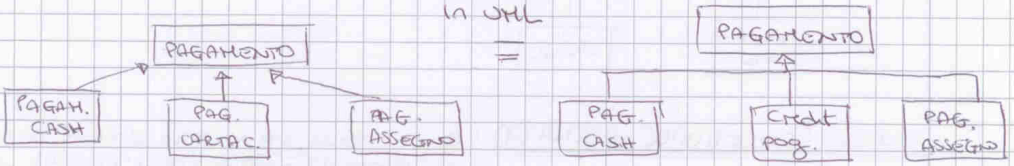
Nell'iterazione 2 [MONOPOLY] l'estensione dei requisiti
- pagamento → effettuato tramite introduzione di un attributo
- caselle speciali → introduzione delle specializzazioni o generalizzazioni.

Iterazione 2 [POS NEXT GEN] estensione dei requisiti
- diverse tipologie di pagamenti → generalizzazione

GENERALIZZAZIONE: è un'attività di astrazione, che porta ad identificare caratteristiche comuni tra concetti
- porta a definire una relazione tra SUPERCLASSE e SOTTOCLASSI

1° generalizzazione ed ereditarietà sono due teorie diverse anche se la notazione in UML è la stessa
due concetti uno è più generale mentre l'altro è più specializzato (rispetto) dell'altro.

In UML =



1) Bisogna capire innanzi tutto quando sia corretto utilizzare una generalizzazione

2) Una volta che lo so applicare correttamente bisogna chiedersi quando sia VERAMENTE UTILE farlo.
- date due classi A e B se la definizione di A non è più generale o comprensiva della definizione di B. => non è esatto dire che B è sottoclasse di A
- inoltre se ci sono istanze di B che non sono anche istanze di A allora non è esatto dire che B è sottoclasse di A (REGOLA È UN)
- REGOLA DEL 100%: si fa riferimento alle caratteristiche strutturali, tutte le caratteristiche sono applicabili anche a ciascuna delle sottoclassi

deve essere verificate tutte e 3 contemporaneamente altrimenti si perde di vista il concetto

può essere utile farlo quando si identifica una partizione di una classe concettuale
PARTIZIONE: (classe è sinonimo di insieme) la partizione di un insieme è un insieme di insiemi i cui elementi sono mutuamente disgiunti e la cui unione forma l'insieme di PARTENZA

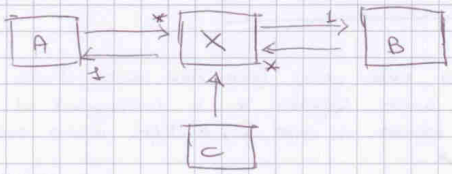
tuttavia la soluzione di detto proposito fa sì che il controller non si debba ricordare solamente i due oggetti A e B ma anche la sequenza! (non è una soluzione ottimale)

SOLUZIONE CABIBBO:

durante la preparazione creo l'oggetto X formo i collegamenti uscenti da X ma non quelli entranti ed inoltre mi ricordo l'oggetto X corrente.
Poi durante la conferma creerò i collegamenti entranti in X da A e B rispettivamente.

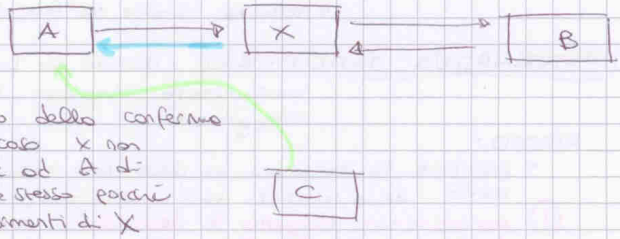
OSS: fare lavoro nella fase di conferma è normale, mentre invece in generale quando devo annullare la cosa deve essere semplice

- il controller può delegare la conferma e dice all'oggetto X, ti confermo
- l'oggetto X quando gli si dice ti confermo deve chiedere ad A di aggiungere X e farà quindi aggiungi (se stesso), e se fosse nella classe di progetto un associazione ad 1 farebbe SET(X) e non add



come fa l'oggetto a confermarsi?
Chiede a tutti gli oggetti di formare un collegamento con se stesso.

e se volessi una situazione del genere?



nel momento della conferma in questo caso X non può chiedere ad A di aggiungere se stesso poiché A non ha riferimenti di X

Quale può essere la soluzione?

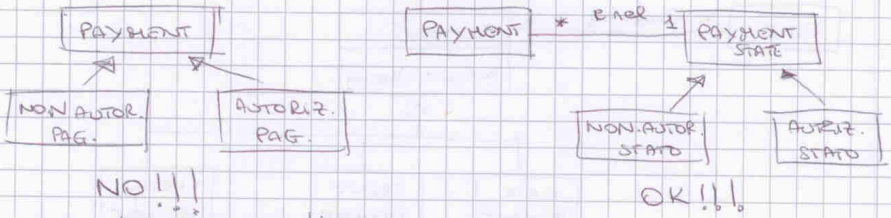
1) O c'è già un accoppiamento tra il controller ed A visibilità
2) se invece non c'è già un accoppiamento e quindi comunque per questo problema devo aggiungere un collegamento di la scelta migliore in questo caso è (anche se non serve) formare il collegamento che va da X ad A (risparmiando così il controllore)

in generale dipende dalla situazione specifica

Analisi e progettazione del software

9/5/13
Lezione XXIII

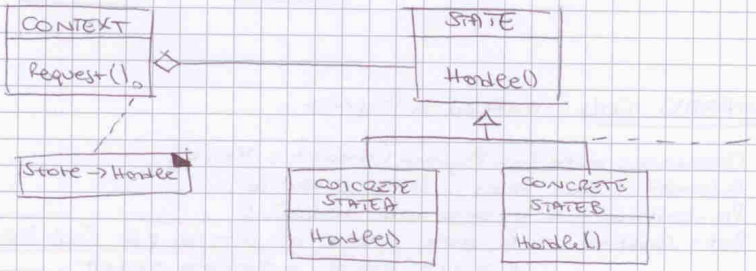
N.B:
Supponiamo che ci sono degli oggetti che hanno uno stato che può cambiare.



NO!!!
perché durante la vita di un oggetto l'oggetto non può cambiare tipo

OK!!!

PATTERN STATE (design): (GOF)



N.B: posso modellare i nuovi con le classi oppure con le associazioni.

13/5/13
Lezione XXIV

NUOVE OP. DI SISTEMA & CONTRATTI

Le estensioni sono modi diversi di eseguire un'operazione, nella seconda ITERAZIONE di PAS introduciamo diversi tipi di pagamento.

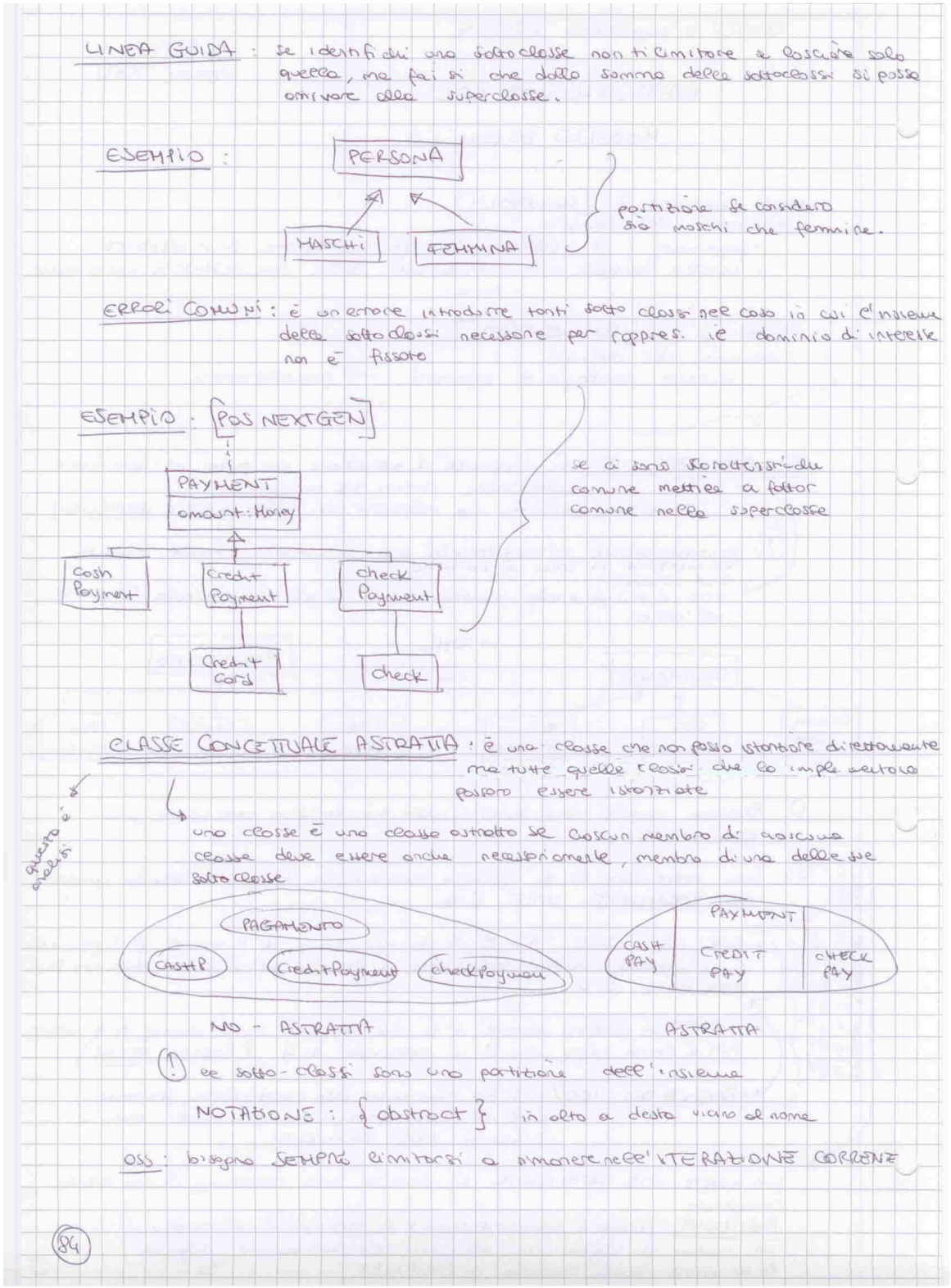
- Le estensioni mi portano ad individuare nuove operazioni di sistema (non le metto in genere nel diagramma di sequenza di sistema)

OSS: dapprima viene esaminato un solo scenario del caso d'uso e una volta modellato si passa alle estensioni

CONTRATTI & GENERALIZZAZIONI:

Se A è una classe astratta è lecito scrivere nei contratti che è stata creata? → assolutamente NO poiché le classi astratte non si possono istanziare, invece si possono istanziare le sottoclassi che le implementano

(!) Il tipo degli oggetti non può mai cambiare, né a livello software né a livello concettuale (per estensione).


LINEA GUIDA : se identifichi una sottoclasse non ti limitare a cercare solo quella, ma fai sì che dalla somma delle sottoclassi si possa arrivare alla superclasse.
ESEMPIO :
PERSONA
MASCHI
FEMMINA
partizione se consideri solo maschi e femmine.
ERRORI COMUNI : è un errore introdurre tante sotto classi nel caso in cui l'insieme delle sottoclassi necessarie per rappres. il dominio di interesse non è fissato
ESEMPIO :
POS NEXTGEN
PAYMENT
amount: Money
Cash Payment
Credit Payment
Check Payment
Credit Card
Check
se ci sono funzionalità che comune mettiamo a fattor comune nella superclasse
CLASSE CONCETTUALE ASTRATTA : è una classe che non posso istanziare direttamente ma tutte quelle classi che da ogni vertice possono essere istanziate
questo è quello
una classe è una classe astratta se ciascun membro di qualsiasi classe deve essere anche necessariamente membro di una delle sue sotto classe
PAGAMENTO
CashP
CreditPayment
checkPayment
NO - ASTRATTA
PAYMENT
CASH PAY
CREDIT PAY
CHECK PAY
ASTRATTA
le sotto-classi sono una partizione dell'insieme
NOTAZIONE : {abstract} in alto a destra vicino al nome
OSS : bisogna SEMPRE limitarsi a rimanere nell'ITERAZIONE CORRENTE
84

Analisi e progettazione software

13/5/13
lezione XXIV

CONTRATTO CO6 : MAKE CHECK PAYMENT

Operazione : makeCheckPayment (driversLicenseNumber)
Riferimenti : Caso d'uso : Effettuare vendita
Pre-condizioni : è in corso una vendita e sono stati inseriti tutti gli articoli.
Post-condizioni :
- È stato creato un checkPayment pmt
- pmt è stato associato con la sale corrente
- è stata creata una Drivers License dl
- gli attributi di dl sono stati inizializzati
- dl è stato associato con pmt
- è stata creata una CheckPaymentApprovalRequest cpr
- pmt è stato associato con cpr
- la sale corrente è stata associata allo STORE come vendita completata.

ULTERIORI PATTERN GRASP

oss : i 5 pattern di cui abbiamo parlato fino ad ora si ispirano al modello di Domini e alcune volte possono darci dei "consigli" sull'assegnazione delle responsabilità in modo errato.

PURE FABRICATION : [PATTERN]

Problema : a quale oggetto deve avere una responsabilità, quando non si vogliono violare Low Coupling, e High Cohesion ma le soluzioni fornite da IE o altri pattern non sono appropriate ?

ES : TE con persistenza dei dati

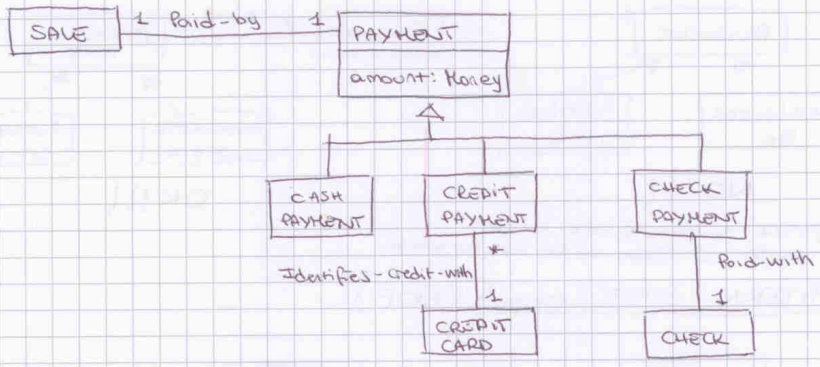
Soluzione : assegna un insieme di responsabilità altamente coesivo ad una classe, che non rappresenta nessun oggetto del dominio ovvero a una classe inventata per sostenere COESIONE ALTA e accoppiamento BASSO.

oss : introdurre un inventario in modo puro e pulito ; e PF lo devo applicare solo se è l'ultimo spiraglio

ESEMPIO : [MONOPOLY]

ie giocatore è accoppiato fortemente con i DADI ; deve tirare i dadi e sommare le singole facce ;
È vero che ie giocatore nella vita reale tira i dadi, ma nel progettare la classe software, questo è diverso quindi la classe giocatore deve avere informazioni coerente riguardanti ie GIOCATORE

ESEMPIO: nel caso di POS è sbagliato cambiare solo il parametro che viene passato (situazione dei diversi tipi di pagamenti), normalmente avremmo delle operazioni di sistema diverse per ciascun tipo di pagamento



CONTRATTO CO4 : makeCashPayment

Operazione: makeCashPayment(amount: Money)
Riferimenti: caso d'uso: Effettua Vendita
Pre-condizioni: è in corso una vendita s
Post-condizioni:
- è stata creata un istanza p di CashPayment
- p.amountTendered è diventato amount
- p è stata associata con la sale corrente s
- la sale corrente s è stata associata con lo store - la vendita è stata aggiunta al registro storico delle vendite completate.

CONTRATTO CO5 : makeCreditPayment

Operazione: makeCreditPayment(CreditAccountNumber, expiryDate)
Riferimenti: caso d'uso: Effettua vendita
Pre-condizione: è in corso una vendita e sono stati inseriti tutti gli articoli
Post-condizioni:
- è stato creato un CreditPayment pmt
- pmt è stato associato con la sale corrente
- è stata creata una CreditCard cc
- gli attributi di cc sono stati inizializzati
- cc è stata associata con pmt
- è stata creata una CreditPaymentApprovalRequest cpr
- pmt è stato associato cpr
- è stata creata una ReceivableEntry re
- re è stato associato all'AccountsReceivable esterno
- la sale corrente è stata associata allo store come vendita completata.

Analisi e progettazione del software

13/5/13
Lezione XXIV

POLIMORFISMO : [PATTERN]

Soluzioni : - quando ho alternative oppure i comportamenti correlati variano con il tipo devono assegnare le responsabilità ai tipi per i quali il comportamento VARIA utilizzando operazioni POLIMORFE → (operazioni che hanno lo stesso nome ma implementate in maniera diversa)
↳ INTERFACCIA o SUPERCLASSE

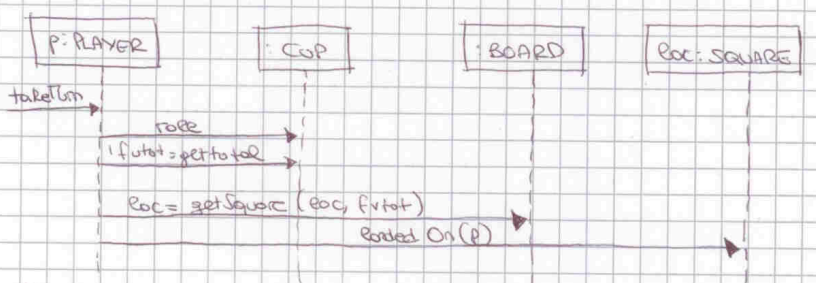
oss : ci sono dei tipi che ~~cambiano~~ controllano le cambiamenti del comportamento
oss : non di cui ma per cui

Corollario : dice che se ci sono alternative che variano in base al tipo non usare istruzioni condizionali ma usa il polimorfismo
(NON TESTARE IL TIPO DEGLI OGGETTI)

- CAMPAGNA ANTI-IF -
ci sono delle situazioni (in cui il comportamento varia con il tipo) e NON bisogna usare l'if

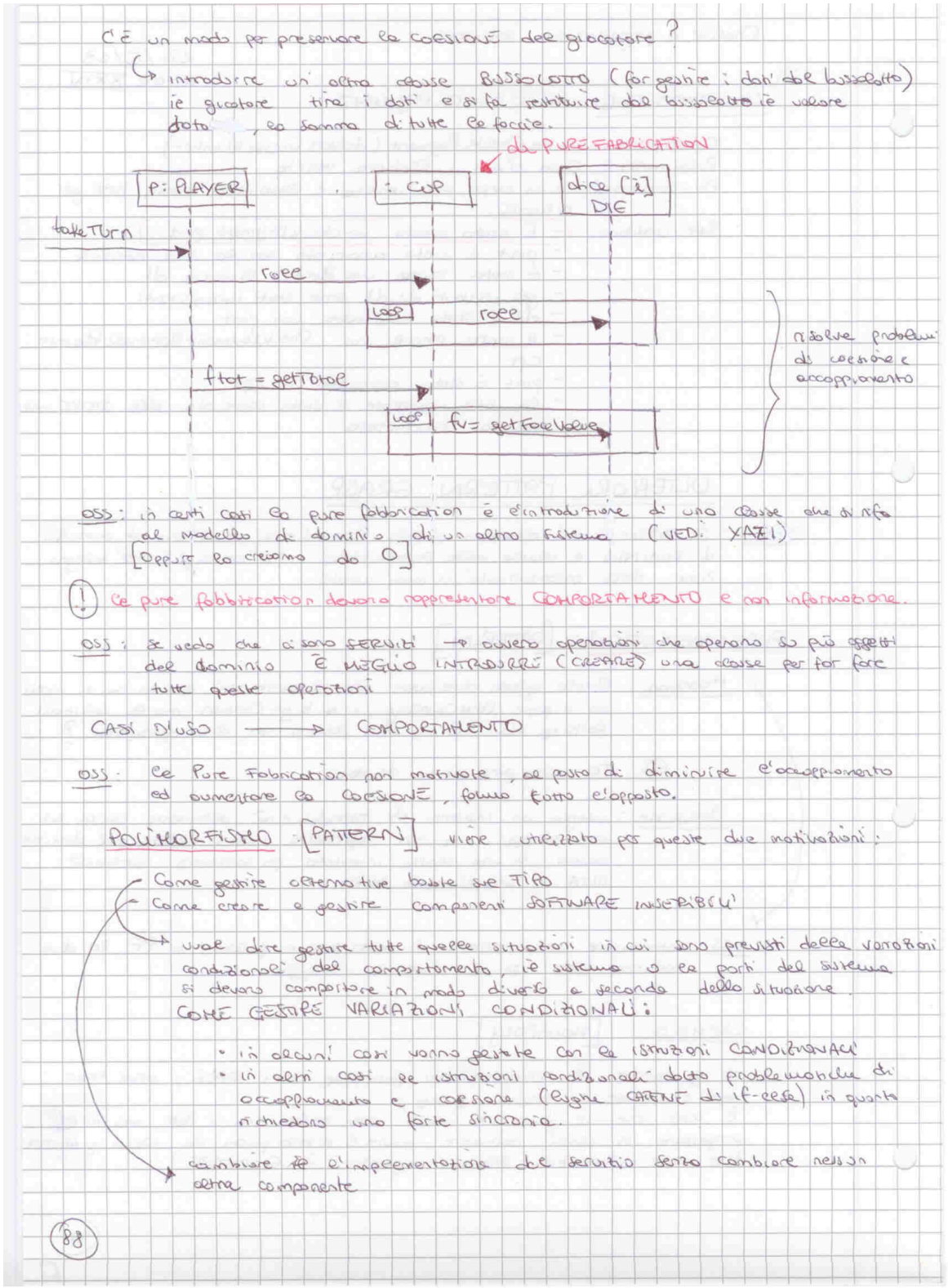
Chiedersi: Quale comportamento varia? Sulla base di quali oggetti il comportamento varia? } con VISIBILITÀ per parametro

Quando utilizziamo un'interfaccia?
quando quella classe non ha dati, non ha variabili d'istanza : infatti questa è la scelta preferita anche se non avrà ovvio in tutti i casi
Quando utilizziamo una classe astratta?
quando la classe ha delle variabili d'istanza.



! devo fare tanti diagrammi di INTERAZIONE quante sono le sottoclassi che implementano l'interfaccia o la classe astratta

ES: è un ERRORE fare cosa deve MONOPOLY assegnare al giocatore la gestione di questo comportamento variabile (dal giocatore)


C'è un modo per preservare la coesione del giocatore?
introdurre un altro classe BUSSOLOTTO (for gestire i dati del bussolotto)
ie giocatore tira i dati e si fa restituire dal bussolotto ie valore dato, ea somma di tutte ea facce.
da PURE FABRICATION
p: PLAYER
: CUP
dice [i] DIE
takeTurn
roee
loop
roee
risolve problemi di coesione e accoppiamento
ftot = getTotroe
loop
fv = getFaceValue
OSS: in certi casi ea pure fabbrication è d'introduzione di una classe che si rifa al modello di dominio di un altro sistema (VED. XAZI)
[Oppure ea creiamo da 0]
ea pure fabbrication devono rappresentare COMPORTAMENTO e non informazione.
OSS: se vedo che ci sono SERVIZI → ovvero operazioni che operano su più oggetti del dominio È MEGLIO INTRODURRE (CREARE) una classe per far fare tutte queste operazioni
CASI D'USO → COMPORTAMENTO
OSS: ea Pure Fabbrication non motivate, ea paura di diminuire l'accoppiamento ed aumentare ea COESIONE, faccio tutto l'opposto.
POLIMORFISMO [PATTERN] viene utilizzato per queste due motivazioni:
Come gestire alternative basate sul TIPO
Come creare e gestire componenti SOFTWARE INTERCAMBIABILI
vuol dire gestire tutte quelle situazioni in cui sono previsti delle variazioni condizionali del comportamento, ie sistema o ea parti del sistema si devono comportare in modo diverso a seconda della situazione
COME GESTIRE VARIAZIONI CONDIZIONALI:
in alcuni casi vanno gestite con ea istruzioni CONDIZIONALI
in altri casi ea istruzioni condizionali dotto problemi di accoppiamento e coesione (eviti CATENE di if-else) in quanto richiedono una forte sincronia.
cambiare ea l'implementazione del servizio senza cambiare nessun altra componente
88

Analisi e progettazione del software
Anno Accademico 2012-2013
Homework 4 – Progettazione

Regole: **SCRIVERE IN MODO LEGGIBILE E NON AMBIGUO**. Scrivere il proprio nome su ciascun foglio utilizzato, in alto a destra. Accanto allo svolgimento di ciascun esercizio o domanda va indicato chiaramente l'esercizio o la domanda a cui lo svolgimento è relativo (ad esempio, Esercizio D1, diagramma delle classi di progetto, oppure Esercizio D2, domanda D2.1). Alle domande in cui è prevista una risposta binaria SI/NO ("è opportuno fare questo?") è necessario scrivere in modo esplicito prima la risposta alla domanda (SI oppure NO) e poi dare una breve motivazione della risposta.

In questo homework si faccia ancora riferimento ai requisiti per il sistema **AcmeArti**, descritti in un documento separato e già utilizzati per gli homework precedenti.

Ipotesi di lavoro, valide per tutti gli esercizi di progettazione.

- In tutti gli esercizi che seguono, si faccia l'ipotesi che il sistema in discussione gestisca i propri dati solo in memoria principale. Si supponga anche che durante il caso d'uso di avviamento vengano creati e caricati in memoria tutti gli oggetti le cui informazioni siano già effettivamente disponibili al momento dell'avviamento.
- Per ciascuna operazione di sistema va creato un diagramma di interazione che descrive l'interazione relativa alla trasformazione (cambiamento di stato) provocata dall'operazione di sistema. Per quanto riguarda invece le relative risposte (interrogazioni) eventualmente restituite dal sistema, se non è richiesto esplicitamente allora non bisogna mostrare nei diagrammi di interazione né il calcolo dei dati da restituire né la loro visualizzazione. Tuttavia, per le risposte del sistema, è comunque necessario verificare che i dati da restituire possano essere (facilmente) calcolati sulla base delle navigabilità tra gli oggetti che sono state progettate (vedi anche il punto successivo).
- **Le soluzioni individuate dovranno essere compatibili (in particolare in termini di visibilità, ovvero di navigabilità delle associazioni) con le realizzazioni di tutti i casi d'uso mostrati (UC1-UC4).**
- **Nei diagrammi di interazione, mostrare IN MODO ESPLICITO: tutti i MESSAGGI scambiati tra oggetti, tutte le CREAZIONI di oggetti e tutte le FORMAZIONI e ROTTURE di collegamenti.**
- Nei diagrammi di interazione, motivare le scelte di progetto fatte indicando i pattern GRASP e GoF applicati.
- Nei diagrammi delle classi di progetto, mostrare: (1) per ciascuna classe: il nome della classe, i nomi dei suoi attributi, i nomi delle sue operazioni; e (2) per ciascuna associazione e ciascuna sua estremità navigabile: la freccia di navigabilità, il nome dell'estremità, la molteplicità e, in caso di associazione navigabile a molti, il tipo di collezione scelta.

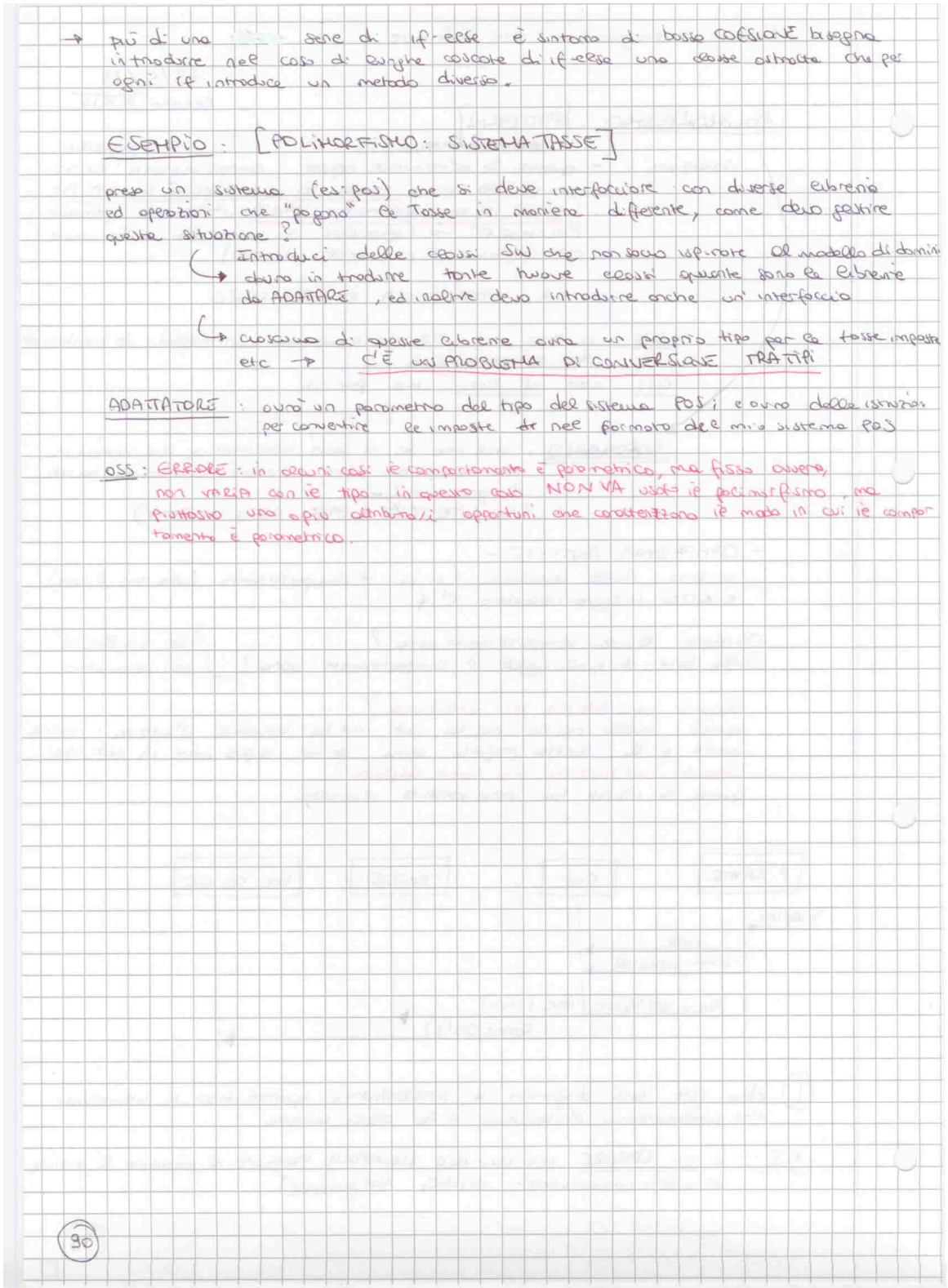
Esercizio D1 (Progettazione)
Fare la progettazione a oggetti per il sistema in discussione, relativamente al caso d'uso UC2 (*Compilazione Piano formativo individuale*), come segue:

- Mostrare i diagrammi di interazione relativi a tutte le operazioni di sistema per il caso d'uso UC2, compresa l'operazione di sistema *annullaInserimentoPFI()* corrispondente all'estensione 4-7a del caso d'uso.
- …
- Mostrare il corrispondente diagramma delle classi di progetto.

Esercizio D2 (Domande)
Relativamente al precedente esercizio D1, rispondere alle seguenti domande (rispondere in modo sintetico, scrivendo circa 3-5 righe per ciascuna domanda):

- …

Esercizio D3 (...)
…



→ più di una serie di if-else è sintomo di bassa COESIONE bisogna introdurre nel caso di cambio cascate di if-else una classe astratta che per ogni if introduce un metodo diverso.

ESEMPIO: [POLIMORFISMO: SISTEMA TASSE]

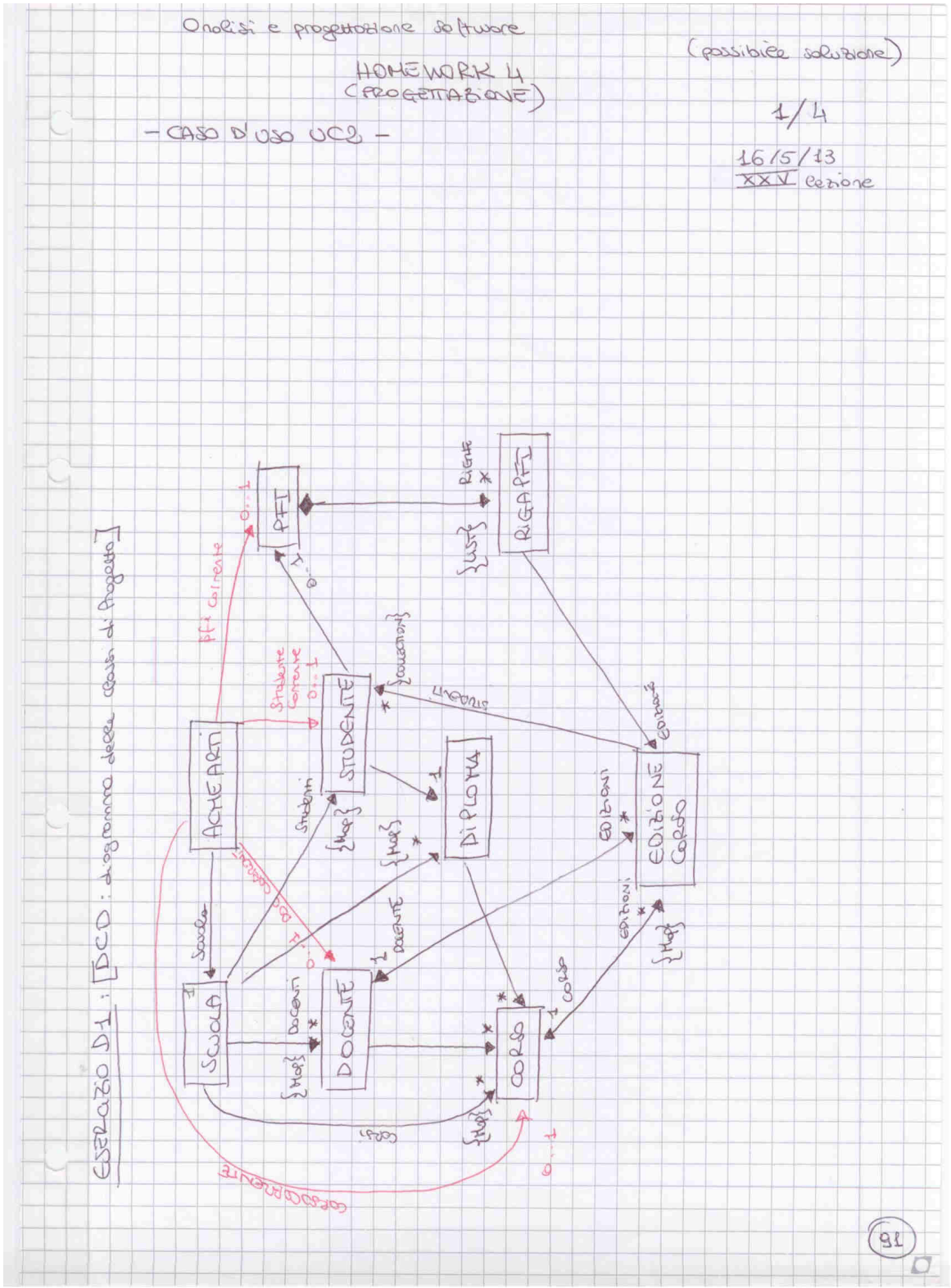
preso un sistema (es: POS) che si deve interfacciare con diverse librerie ed operazioni che "pagano" le Tasse in maniera differente, come devo gestire questa situazione?

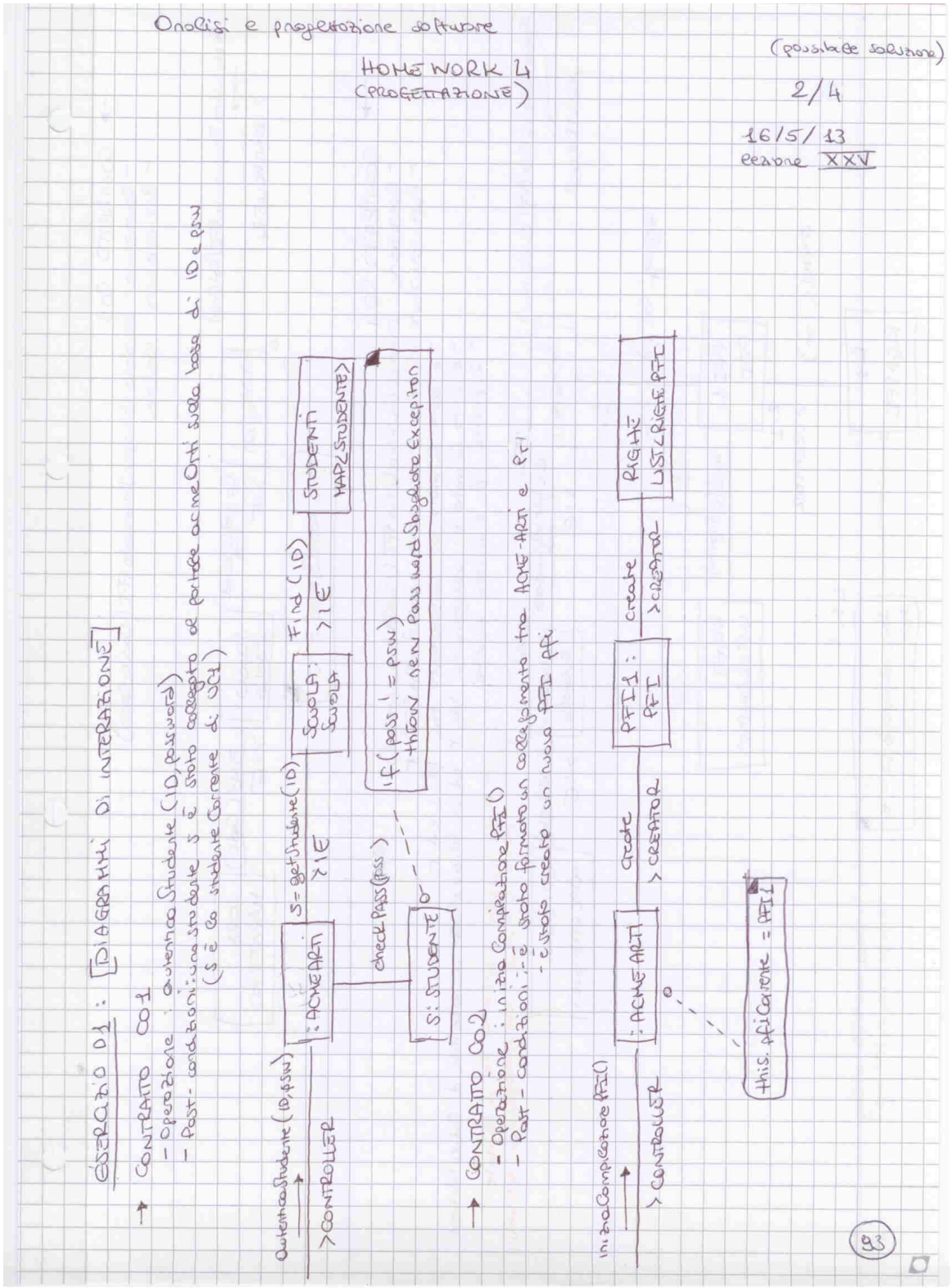
↳ introduci delle classi SW che non sono supportate dal modello di dominio devo introdurre tante nuove classi quante sono le librerie da ADATTARE, ed inoltre devo introdurre anche un'interfaccia

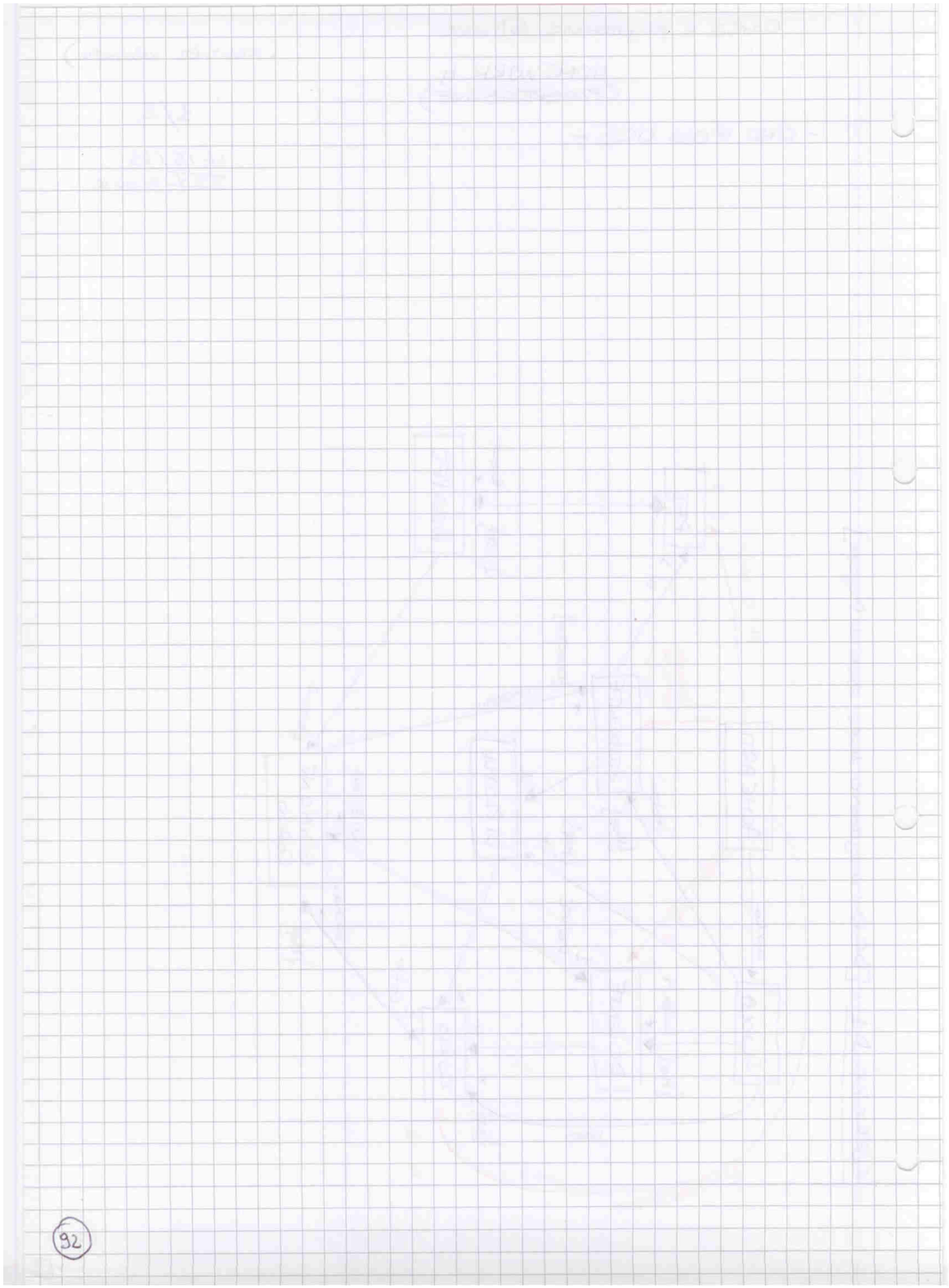
↳ ciascuna di queste librerie avrà un proprio tipo per la tasse importo etc → C'È UN PROBLEMA DI CONVERSIONE TRA TIPI

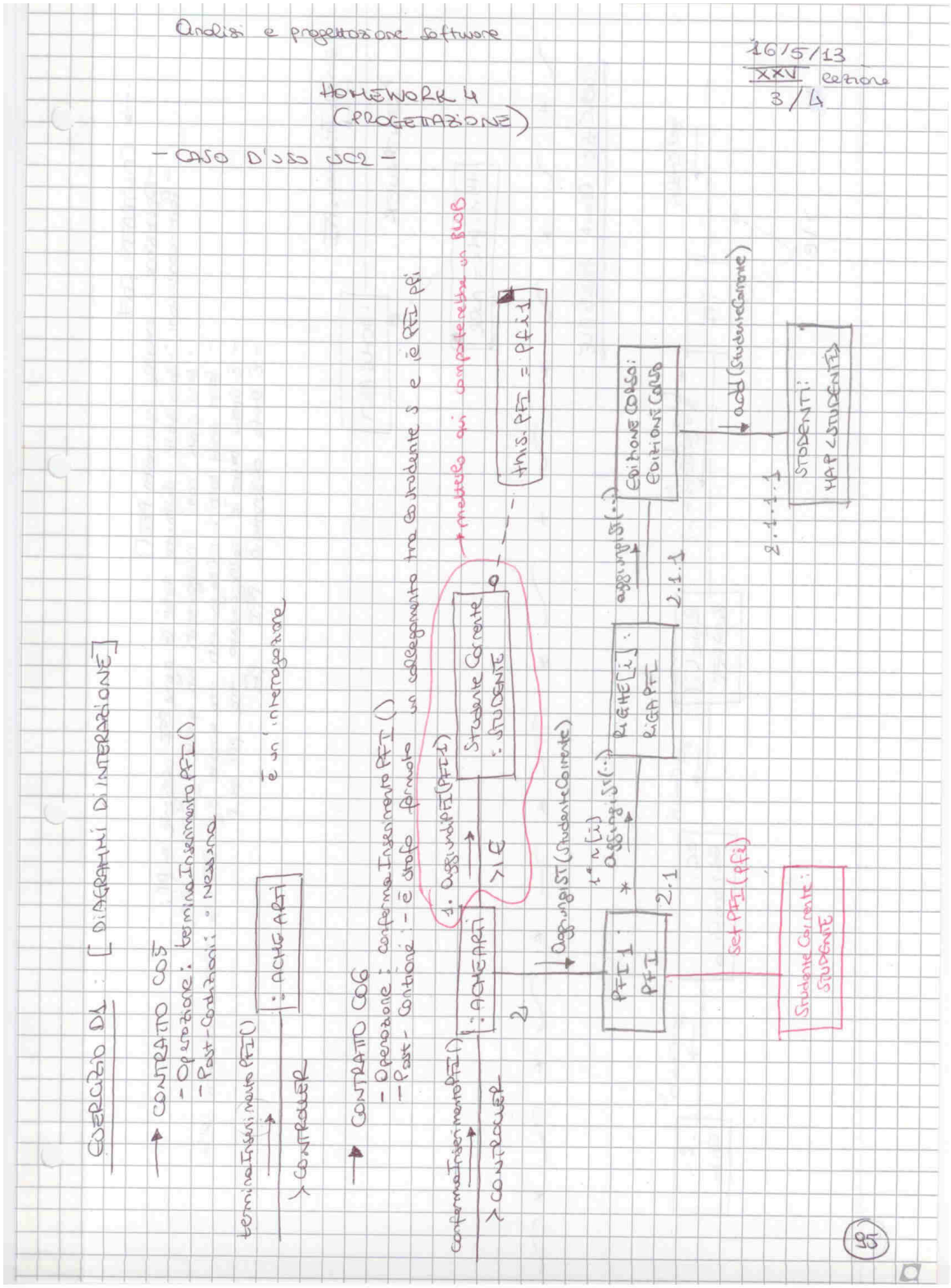
ADATTATORE: avrà un parametro del tipo del sistema POS; e avrà delle istanze per convertire le imposte ~~di~~ nel formato del mio sistema POS

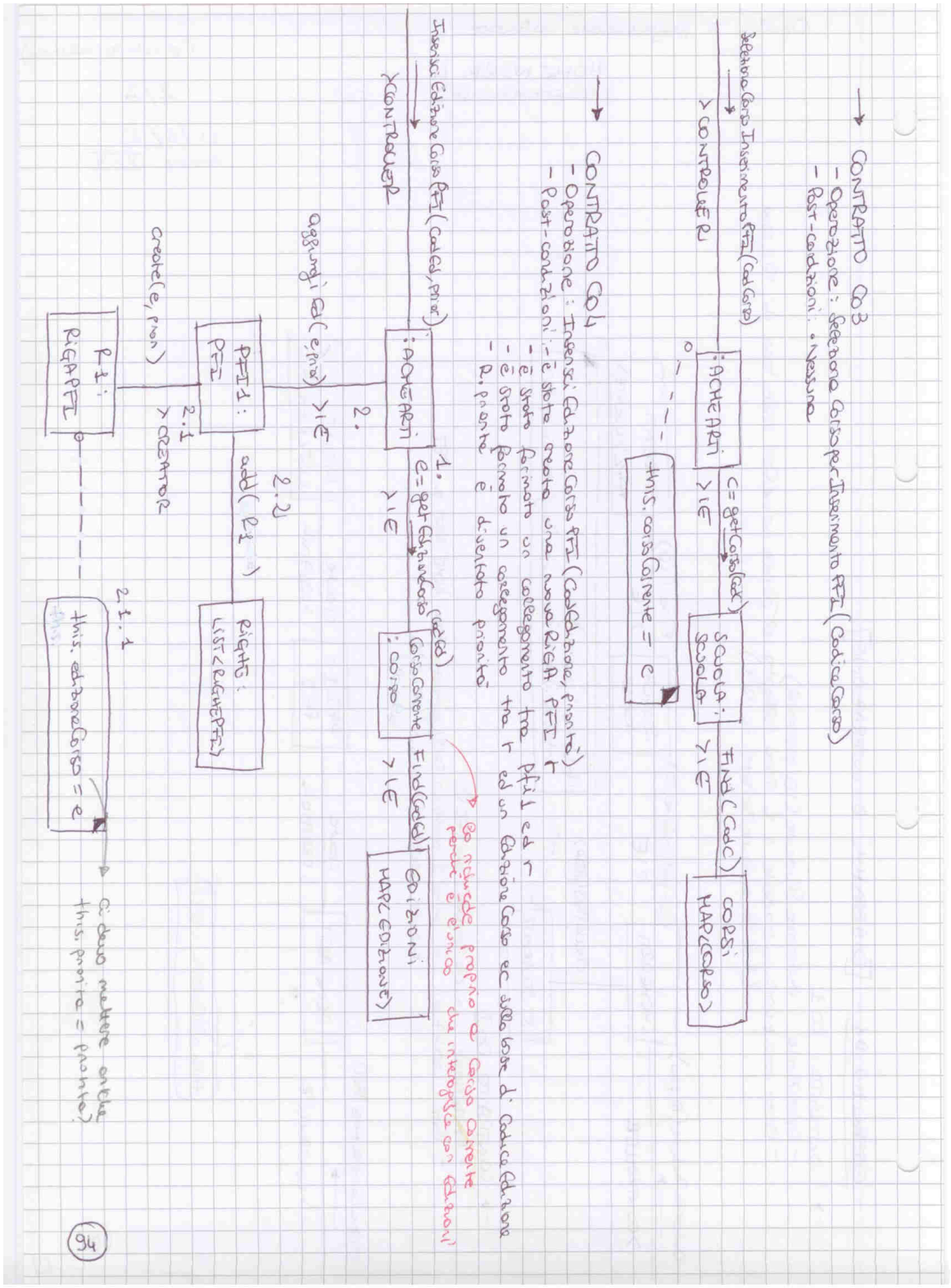
OSS: ERRORE: in alcuni casi il comportamento è parametrico, ma fisso ovvero, non varia con il tipo in questo caso NON VA usato il polimorfismo, ma piuttosto uno spio attributi/i opportuni che caratterizzano il modo in cui il comportamento è parametrico.

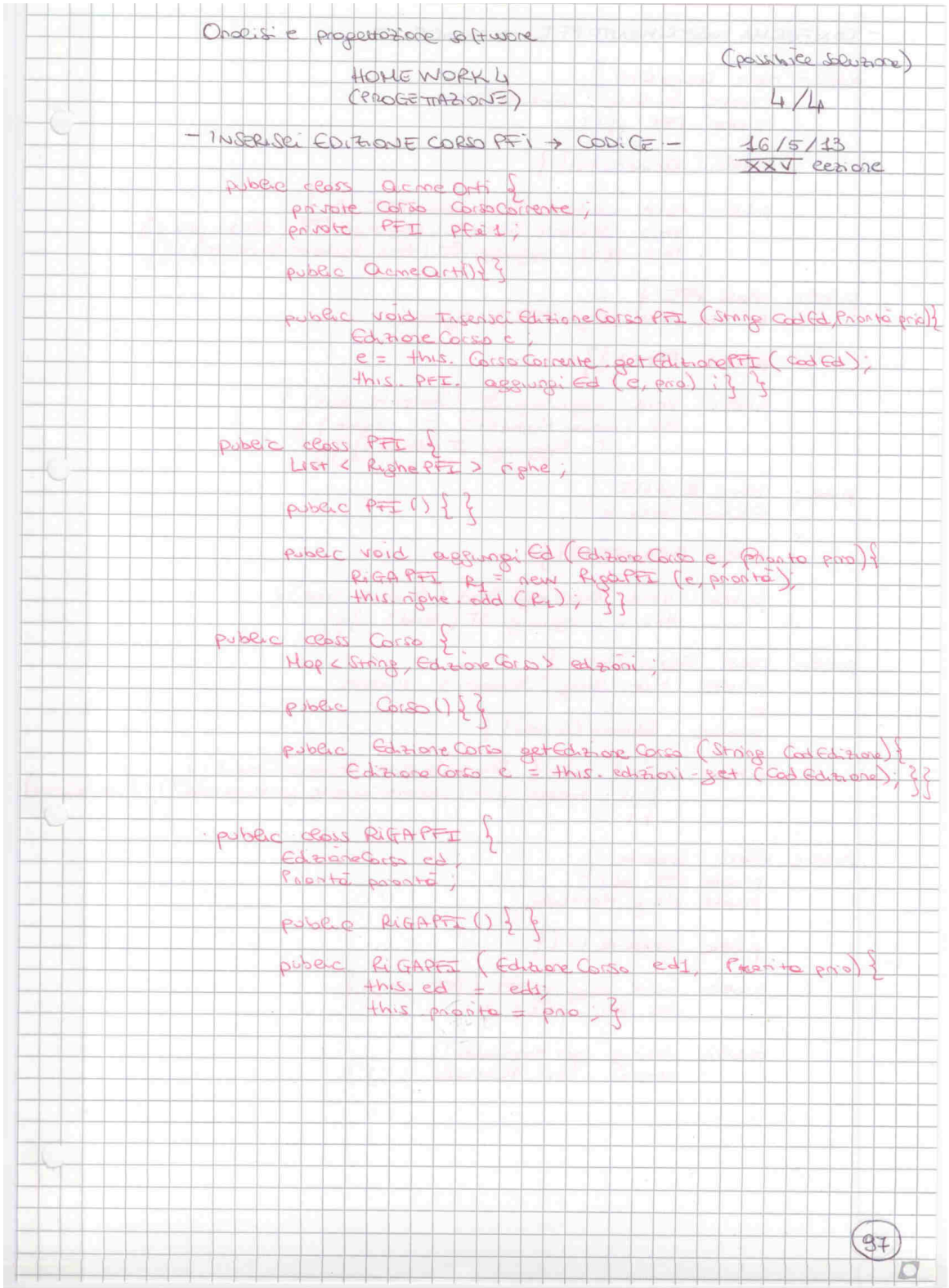

Analisi e progettazione software
(possibile soluzione)
HOMEWORK 4
(PROGETTAZIONE)
- CASO D'USO UC2 -
1/4
16/5/13
XXV lezione
ESERCIZIO D1: [DCD: diagramma delle classi di progetto]
PFI
RIGA PFI
STUDENTE
ACHE ARTI
DIPLOMA
EDIZIONE CORSO
SCUOLA
DOCENTE
CORSO
PFI del mese
studente corrente
corso corrente
docente corrente
edizione corrente
{ordered}
{unique}
{xor}
{or}
91


Analisi e progettazione software
HOME WORK 4
(PROGETTAZIONE)
(possibile soluzione)
2/4
16/5/13
Lezione XXV
ESERCIZIO 01: [DIAGRAMMI DI INTERAZIONE]
CONTRATTO CO1
- Operazione: autenticaStudente(ID,password)
- Post-condizioni: uno studente s è stato collegato al parser acme.Ott sulla base di ID e psw
(s è lo studente corrente di SCU)
autenticaStudente(ID,psw)
: AcmeAPI
s = getStudente(ID)
: Scuola
findById(ID)
STUDENTI
MAP<STUDENTE>
if(pass != psw)
throw new PassWordSbagliataException
check PASS(pass)
s: STUDENTE
CONTRATTO CO2
- Operazione: initAcmeCompleteAPI()
- Post-condizioni: è stato formato un collegamento tra AcmeAPI e API
è stato creato un nuovo API api
initAcmeCompleteAPI()
: AcmeAPI
create
PAPI: API
create
RIGHE
LIST<RIGHE API>
this.APIcorrente = API


92


Analisi e progettazione software
HOMEWORK 4
(PROGETTAZIONE)
- CASO D'USO UC2 -
16/5/13
XXV lezione
3/4
ESERCIZIO DI [DIAGRAMMI DI INTERAZIONE]
CONTRATTO COS
- Operazione: terminaInserimentoPFI()
- Post-condizioni: - Nessuna
terminaInserimentoPFI()
: ACHEAPFI
è un'interrogazione
CONTROLLER
CONTRATTO COG
- Operazione: confermaInserimentoPFI()
- Post-condizioni: - è stato formato un collegamento tra Studente S e le PFI pfi
confermaInserimentoPFI()
: ACHEAPFI
CONTROLLER
1. aggiungiPFI(PFI1)
Studente Corrente
: STUDENTE
>1 C
2
aggiungiPFI(studenteCorrente)
PFI 1
PFI
2.1
1.* n [i] aggiungiPFI(...)
PIGHETE[i]:
PIGA PFI
aggiungiPFI(...)
2.1.1
Edizione Corso:
Edizione Corso
2.1.1.1
ACHE (studenteCorrente)
STUDENTI:
MAP <STUDENTE>
Set PFI (pfi)
Studente Corrente:
STUDENTE
95


CONTRATTO CO3
- Operazione : rimuoviCorsoPerInsegnamentoPTL(codiceCorso)
- Post-condizioni : • nessuno
rimuoviCorsoPerInsegnamentoPTL(codCorso)
:SCHEDARIO
SCUOLA :
Scuola
this.corsoCorrente = c
CONTRATTO CO4
- Operazione : inserisciEdizioneCorsoPTL(codEdizione, progetto)
- Post-condizioni : • è stata creata una nuova PTL t
• è stata formata un collegamento tra t ed un Edizione Corso ec sulla base di codice Edizione
• è stato formato un collegamento tra t ed un Docente d
• P.progetto è diventato progetto
Inserisci Edizione Corso PTL(codEd, prog)
:SCHEDARIO
:CORSO
PTL :
PTL
P :
RIGHTPTL
this.edizioneCorso = e
Edizioni
Docente corrente
Rights
LISTA RIGHTPTL
Il docente proprio di questo corso interagisce con questi



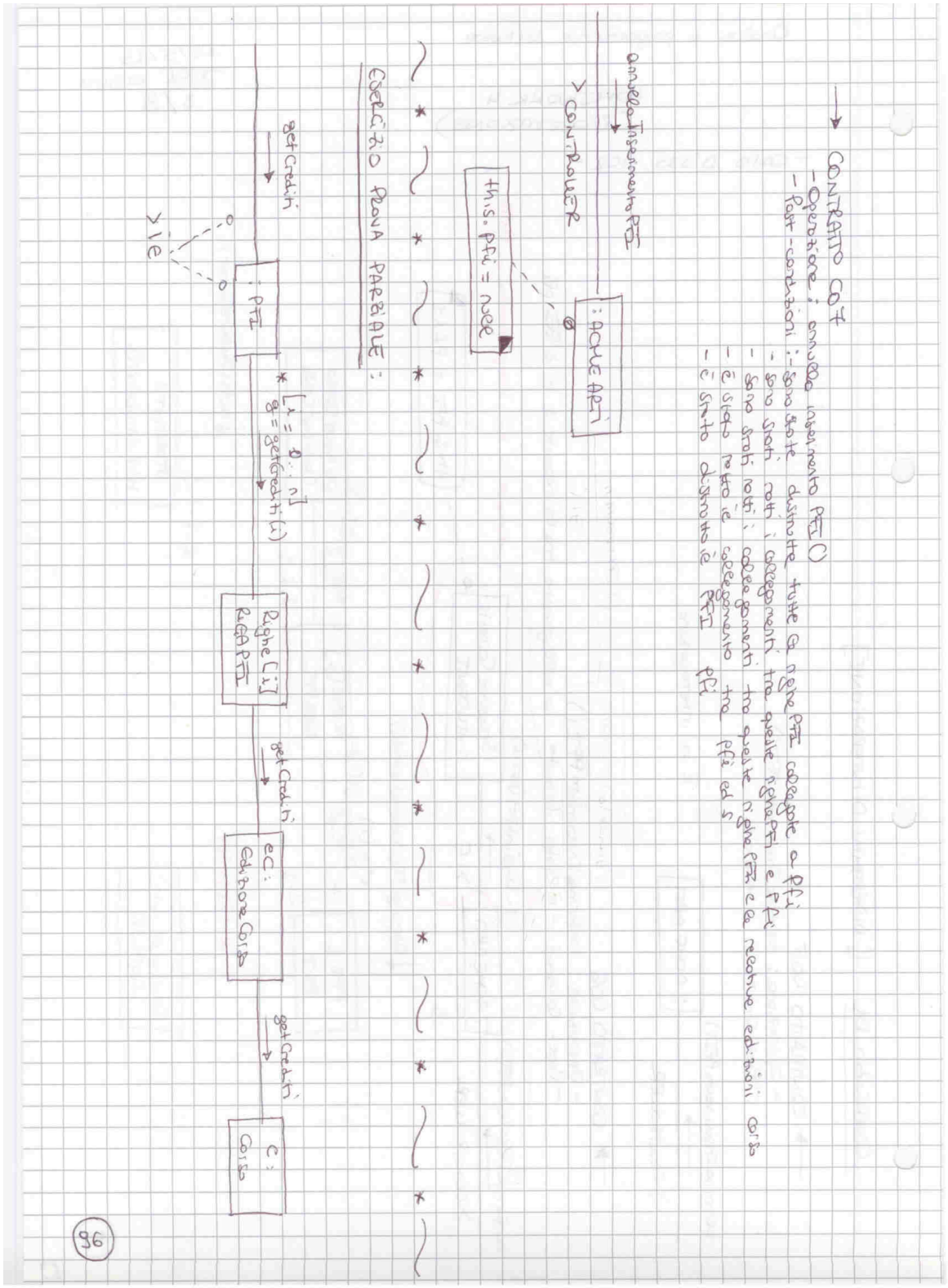
Analisi e progettazione software (possibile soluzione)

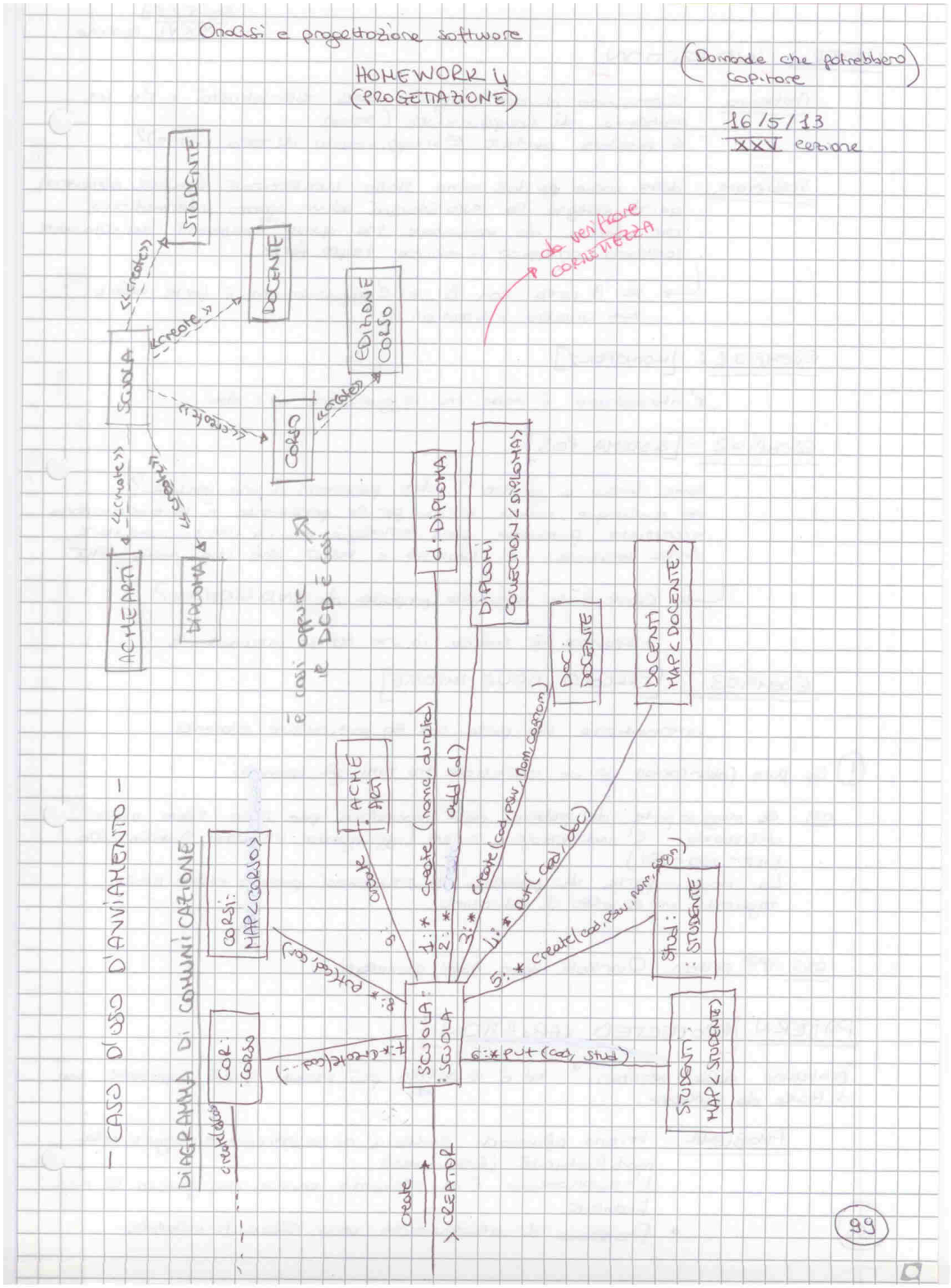
HOME WORK 4 4/4
(PROGETTAZIONE)

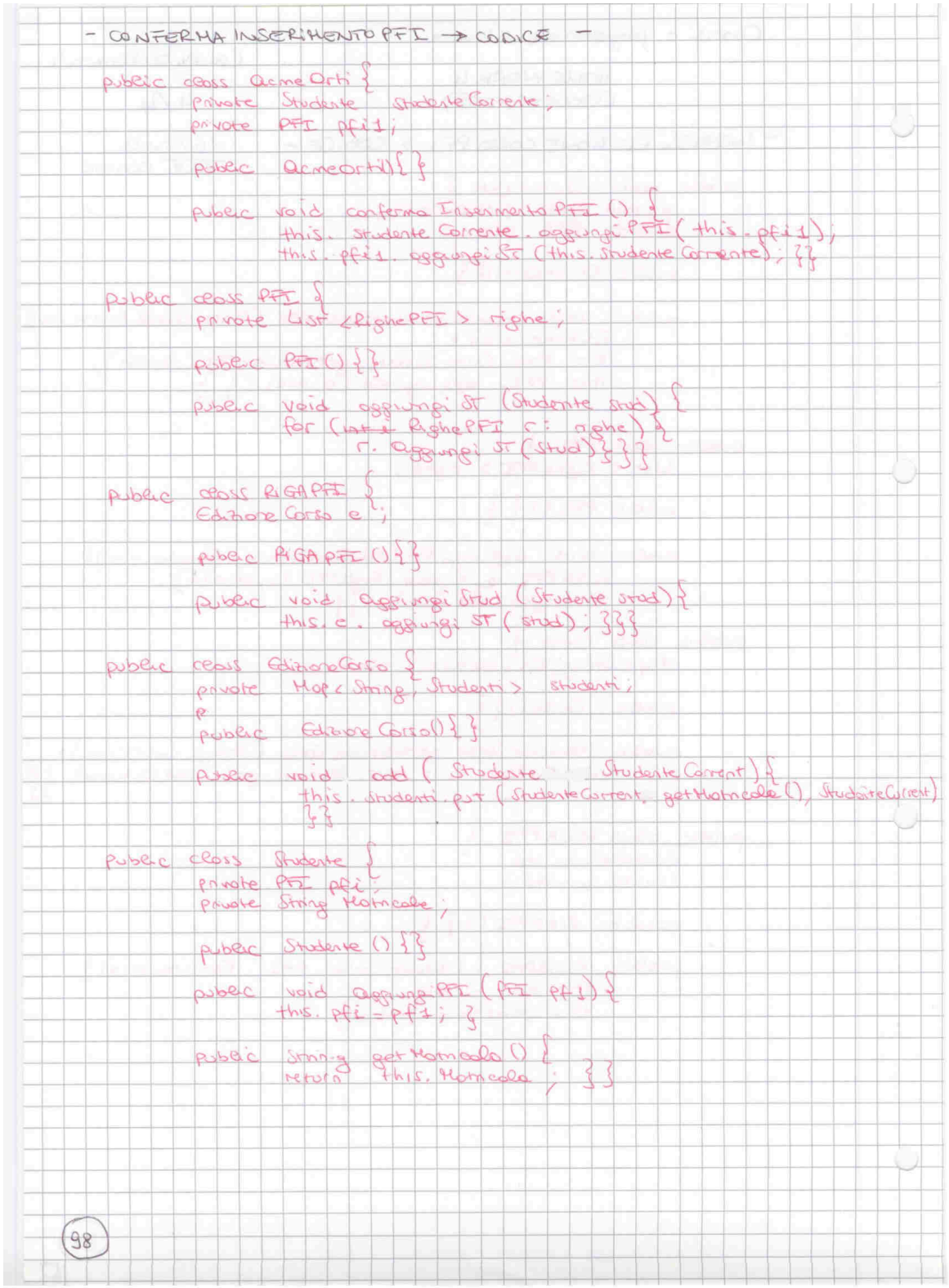
- INSERISCI EDIZIONE CORSO PFI → CODICE - 16/5/13
XXV esercizio

```
public class AcmeArti {
    private Corso CorsoCorrente;
    private PFI pfi1;

    public AcmeArti(){}

    public void InserisciEdizioneCorsoPFI (String CodEd, Priorità prio){
        EdizioneCorso e;
        e = this.CorsoCorrente.getEdizionePFI (CodEd);
        this.PFI.aggiungiEd (e, prio); }}

public class PFI {
    List < RighePFI > righe;

    public PFI(){}

    public void aggiungiEd (EdizioneCorso e, Priorità prio){
        RigaPFI R1 = new RigaPFI (e, priorità);
        this.righe.add (R1); }}

public class Corso {
    Map< String, EdizioneCorso > edizioni;

    public Corso(){}

    public EdizioneCorso getEdizioneCorso (String CodEdizione){
        EdizioneCorso e = this.edizioni.get (CodEdizione); }}

public class RigaPFI {
    EdizioneCorso ed;
    Priorità priorità;

    public RigaPFI(){}

    public RigaPFI (EdizioneCorso ed1, Priorità prio){
        this.ed = ed1;
        this.priorità = prio; }
```

97


CONTRATTO CO1
- Operazione: annullaInserimentoPTL()
- Post-condizioni: - sono state distrutte tutte le righe PTL collegate a PTL
- sono stati rotti i collegamenti tra queste righePTL e PTL
- sono stati rotti i collegamenti tra queste righePTL e PTL
- è stato posto a null il collegamento tra PTL ed S
- è stato distrutto il collegamento PTL
annullaInserimentoPTL()
: CONTROLLER
: ACME API
this.ptl = null
ESERCIZIO PROVA PARZIALE:
getCrediti()
: PTL
vle
righePTL = getRighePTL(i)
Righe[i] RigaPTL
getCrediti()
EC: Edizione Gris
getCrediti()
C: Corso
96


Analisi e progettazione software
HOMEWORK 4
(PROGETTAZIONE)
(Domande che potrebbero capitare)
16/5/13
XXV sessione
da verificare correttezza
STUDENTE
DOCENTE
EDIZIONE CORSO
SCUOLA
CORSO
ATTIVITÀ
DIPLOMA
«create»
d: DIPLOMA
DIPLOMI: COLLECTION<DIPLOMA>
DOCENTI: MAP<DOCENTE>
DOC: DOCENTE
ATTIVITÀ: ATTI
CORSI: MAP<CORSO>
COR: CORSO
STUD: STUDENTE
STUDENTI: MAP<STUDENTE>
SCUOLA: SCUOLA
CREATOR
1: * create (nome, durata)
2: * add (d)
3: * create (cod, pers, nom, cognom)
4: * create (cod, pers, nome, cognom)
5: * put (cod, doc)
6: * put (cod, stud)
- CASO D'USO D'AVVIAMENTO -
DIAGRAMMA DI COMUNICAZIONE
99



- CONFERMA INSERIMENTO PFI → CODICE -

```
pubeic ceass AcmeOrti {
        private Studente studenteCorrente;
        private PFI pfi1;

        pubeic AcmeOrti(){}

        pubeic void confermaInserimentoPFI () {
        this. studenteCorrente. aggiungiPFI (this. pfi1);
        this. pfi1. aggiungiST (this. studenteCorrente); }}

pubeic ceass PFI {
        private List <RighePFI> righe;

        pubeic PFI () {}

        pubeic void aggiungiST (Studente stud) {
        for (int i: righe) {
                r. aggiungiST (stud)}}}

pubeic ceass RIGAPFI {
        EdizioneCorso e;

        pubeic RIGAPFI (){}

        pubeic void aggiungiStud (Studente stud){
        this. e. aggiungiST (stud); }}}

pubeic ceass EdizioneCorso {
        private Map< String, Studenti > studenti;
        P
        pubeic EdizioneCorso(){}

        pubeic void add ( Studente studenteCorrent){
        this. studenti. put (studenteCorrent. getMatricola (), studenteCurrent)
        }}

pubeic ceass Studente {
        private PFI pfi;
        private String Matricola;

        pubeic Studente (){}

        pubeic void aggiungiPFI (PFI pfi1) {
        this. pfi = pfi1; }

        pubeic String getMatricola () {
        return this. Matricola; }}
```

Analisi e progettazione del software

20/5/13
XXVI lezione

PATTERN [PROTECTED VARIATION]

PROBLEMA: come fare in modo per progettare sotto altri sistemi in modo tale che le variazioni o oggetti di sotto sistemi - sistemi, non abbiano un impatto indesiderato su gli altri

SOLUZIONE: identifica i punti per cui sono previste variazioni da instabilità e poi assegna delle responsabilità per creare un'interfaccia stabile intorno a questi elementi (ad esempio utilizzando il polimorfismo)

- chiediti in quei punti sono presenti variazioni di stabilità che potrebbero avere un effetto indesiderato su altri elementi.
- poi fai sì che l'interfaccia intorno a questi oggetti sia stabile

ES: [CALCOLO DELLE IMPOSTE]
supponiamo che nel mercato ci siano 5 calcolatori delle imposte, il risultato definito dal polimorfismo è di definire 5 classi che sono adattatori nei confronti di queste librerie

Supponiamo di avere una 6ª libreria che si chiama Open Source e vantaggioso da utilizzare (gratis) → introduce variazioni non tanto nei requisiti, ma nel contesto in cui bisogna utilizzarla.

posso usufruire di questa libreria scrivendo poche righe di una classe adattatore?

↳ per ciascun tipo di calcolatore una classe adattatore che implementa l'operazione definita nell'interfaccia.
Se mercato arriva una nuova libreria → creo un nuovo adattatore

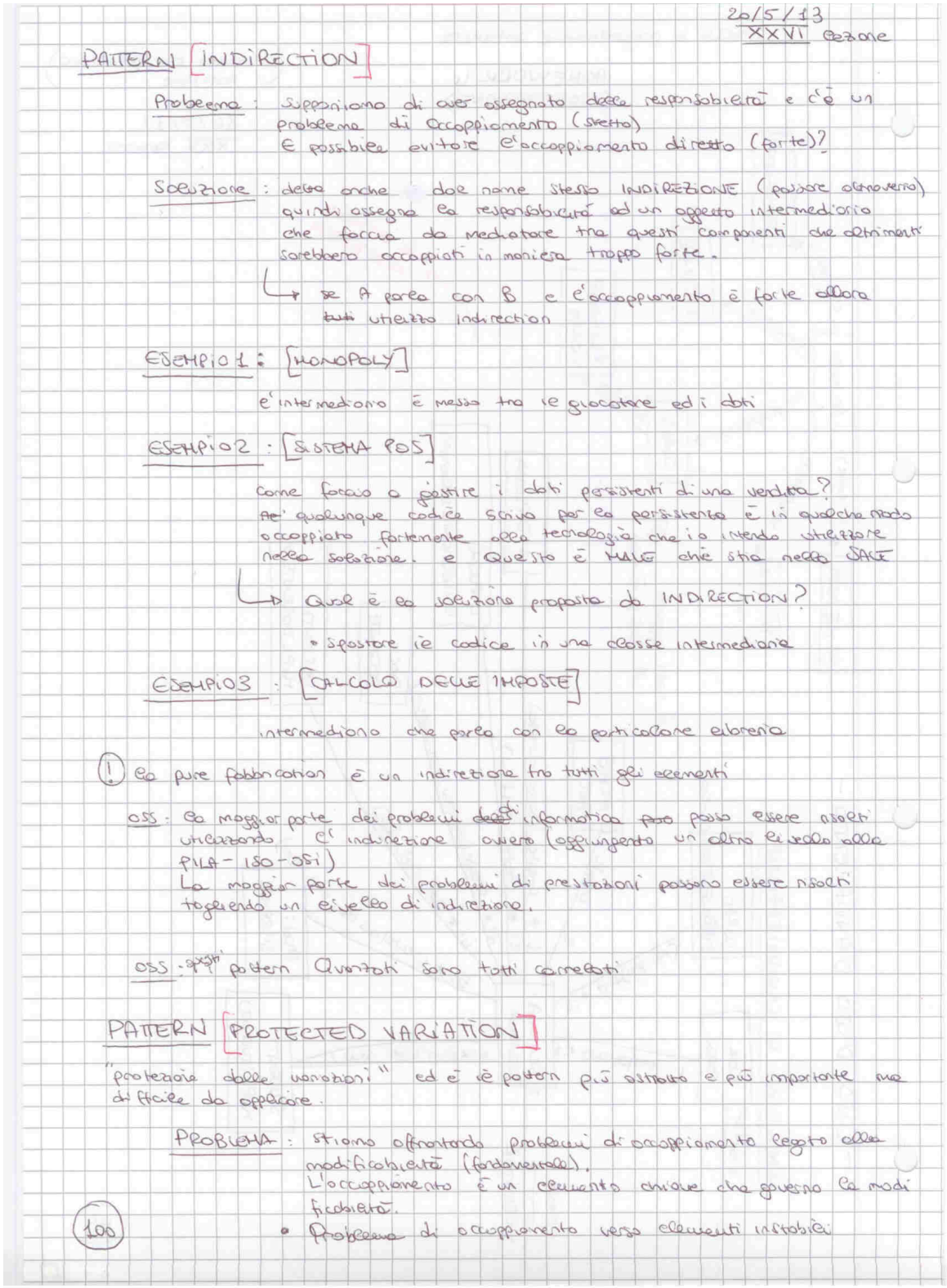
ES: PV va applicato con cautela perché non sempre occorre proteggersi, perché se l'evento negativo non si verifica ha solo sprecato energia

ES: è simile a quando una persona si assicura, e sta pagando quindi un'assicurazione, che maledirà il momento in cui pago ma sono contento se si verifica un incidente perché i soldi non usciranno dalle sue tasche (ANALOGIA CON PV)

! definire delle interfacce è un modo per definire dei punti stabili intorno a cose che possono cambiare

OSS: anche il polimorfismo è una forma di PV

PROGETTAZIONE GUIDATA AI DATI: spostare certi aspetti relativi alla modifica del comportamento del sistema al di fuori del codice → affinché non devo cambiare il codice ma solo un file di configurazione.



26/5/13
XXVI lezione

PATTERN [INDIRECTION]

Problema: Supponiamo di aver assegnato delle responsabilità e c'è un problema di accoppiamento (stretto)
È possibile evitare l'accoppiamento diretto (forte)?

Soluzione: deleg anche due nome stesso INDIREZIONE (passare altrove) quindi assegno la responsabilità ad un oggetto intermediario che faccia da mediatore tra questi componenti che altrimenti sarebbero accoppiati in maniera troppo forte.

↳ se A parla con B e c'è accoppiamento è forte allora utilizzo indirection

ESEMPIO 1: [MONOPOLY]

è intermediario è messo tra le giocatore ed i dati

ESEMPIO 2: [SISTEMA POS]

Come faccio a gestire i dati persistenti di una vendita?
Ad qualunque codice scrivo per la persistenza è in qualche modo accoppiato fortemente alla tecnologia che io intendo utilizzare nella soluzione. e Questo è MALE che sta nella DATE

↳ Qual è la soluzione proposta da INDIRECTION?

- Spostare il codice in una classe intermediaria

ESEMPIO 3: [CALCOLO DELLE IMPOSTE]

intermediario che parla con la particolare elaborata

(!) La pure fabrication è un indirezione tra tutti gli elementi

OSS: La maggior parte dei problemi dell'informatica può posso essere risolti utilizzando l'indirezione ovvero (aggiungendo un altro livello alla PILA - ISO - OSI)
La maggior parte dei problemi di prestazioni possono essere risolti togliendo un livello di indirezione.

OSS: gli pattern Quantati sono tutti corretti

PATTERN [PROTECTED VARIATION]

"protezione delle variazioni" ed è il pattern più astratto e più importante ma difficile da applicare.

PROBLEMA: stiamo affrontando problemi di accoppiamento legato alla modificabilità (fondamentale).
L'accoppiamento è un elemento chiave che governa la modificabilità.

- Problema di accoppiamento verso elementi instabili

100

Analisi e progettazione del software

20/5/13
XXVI Lezione

PATTERN [ADAPTER]:

SOLUZIONE: usando un intermediario tra il client e il servizio che ha lo scopo di convertire l'interfaccia del servizio, in un altra interfaccia gradita dal client sulla base del oggetto inserito

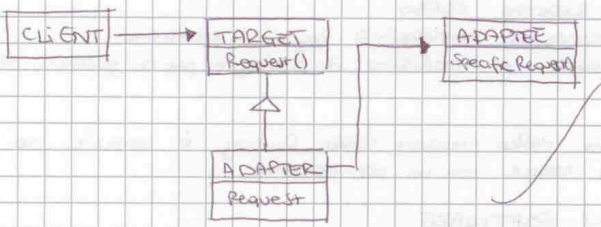
adapter consente a classi diverse di operare insieme quando ciò non sarebbe altrimenti possibile a causa di interfacce incompatibili

- Bisogna chiedersi qual è l'interfaccia del servizio che è gradita dal client?

ES: SIST. [POS NEXT GEN]
il register vorrebbe fare delle richieste del tipo

```
taxLineItem ms = taxCalculator.getTaxes(sale);
```

- il metodo dovrà effettuare una conversione dei parametri dal formato del client al formato del server
- l'uso effettivo del servizio
- il risultato poi verrà riconvertito nel formato gradito da POS



ADAPTER VS GRASP
- è basata sull'uso di un oggetto INDIRECTION
- applica interfacce e polimorfismo
- sostiene PROTECTED VARIATIONS rispetto al cambiamento di interfacce esterne

OSS: noi stiamo supponendo che il client chiede il servizio dopo che è stato selezionato il servizio giusto

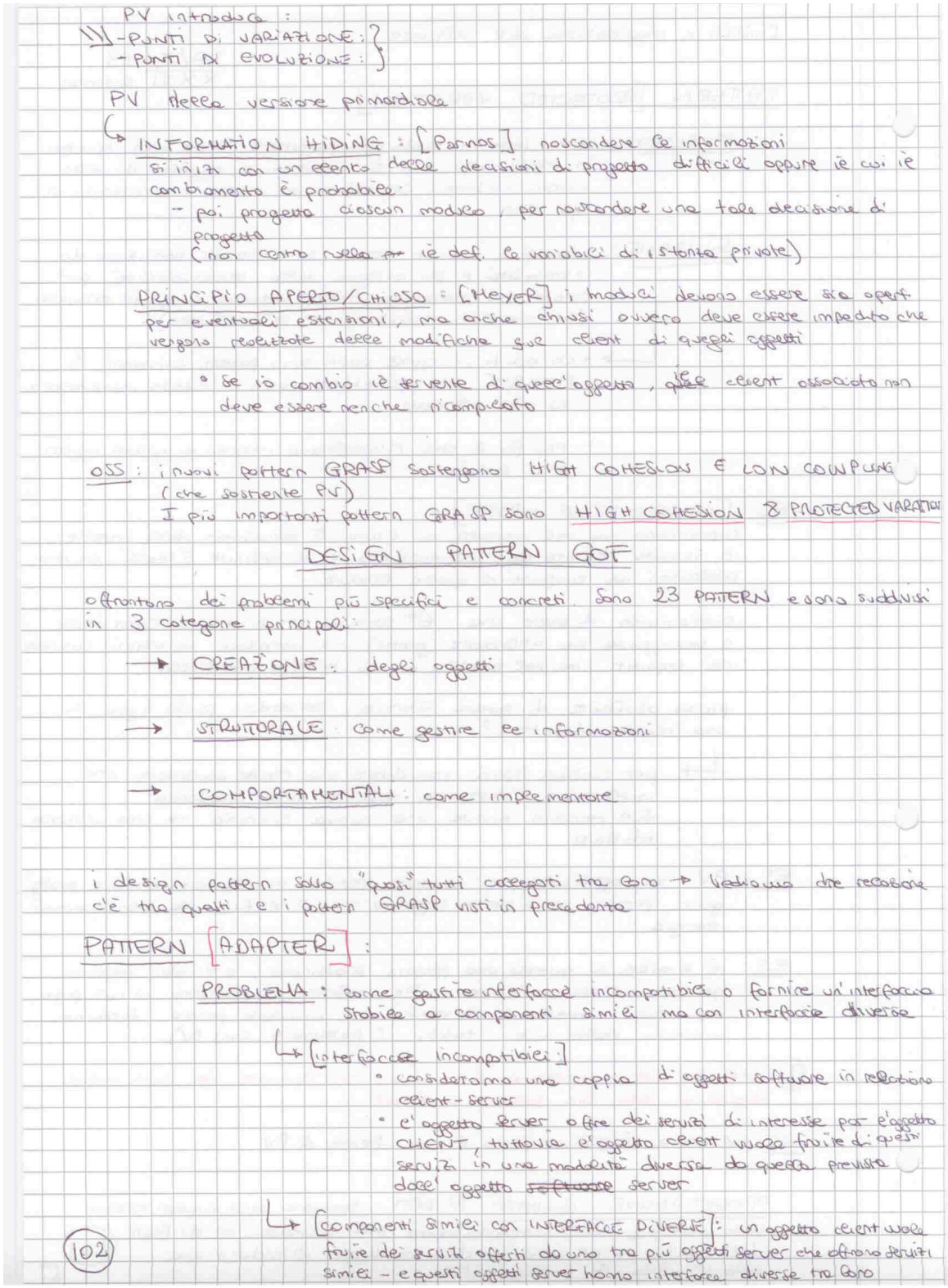
→ Chi e come vengono creati gli oggetti ADATTATORE?

- uno potrebbe dire uso creator, ma non va bene perché creator fa nfai agli oggetti di dominio (in questo caso stiamo considerando ex (PURE FABRICATION))

PATTERN (CONCRETE) [FACTORY]: che ha responsabilità di creare un oggetto.

PROBLEMA: chi deve essere responsabile della creazione di oggetti quando ci sono considerazioni speciali?

Es: una logica di creazione, quando si desidera separare la responsabilità di creazione per una coesione migliore



PV introduce:
- PUNTI DI VARIAZIONE;
- PUNTI DI EVOLUZIONE;

PV definisce versione primordiale

↳ INFORMATION HIDING: [Parnas] nascondere le informazioni
Si inizia con un elenco delle decisioni di progetto difficili oppure le cui il cambiamento è probabile.
- poi progetto ciascun modulo, per nascondere una tale decisione di progetto
(non centro nella ~~~~ le def. le variabili di istanza private)

PRINCIPIO APERTO/CHIUSO: [Meyer] i moduli devono essere sia aperti per eventuali estensioni, ma anche chiusi ovvero deve essere impedito che vengano realizzate delle modifiche sui client di quegli oggetti

- Se io cambio le ~~~~ di quell'oggetto, ~~~~ client associato non deve essere anche ricompilato

OSS: i nuovi pattern GRASP sostengono HIGH COHESION E LOW COUPLING (che sostiene PV)
I più importanti pattern GRASP sono HIGH COHESION & PROTECTED VARIATION

DESIGN PATTERN GOF

affrontano dei problemi più specifici e concreti. Sono 23 PATTERN e sono suddivisi in 3 categorie principali:

→ CREAZIONE: degli oggetti

→ STRUTTURALE come gestire le informazioni

→ COMPORTAMENTALI: come implementare

i design pattern sono "quasi" tutti collegati tra loro → bisogna che relazione c'è tra quelli e i pattern GRASP visti in precedenza

PATTERN [ADAPTER]:

PROBLEMA: come gestire interfacce incompatibili o fornire un'interfaccia stabile a componenti simili ma con interfacce diverse

↳ [interfacce incompatibili]
- consideriamo una coppia di oggetti software in relazione client-server
- l'oggetto server offre dei servizi di interesse per l'oggetto CLIENT, tuttavia l'oggetto client vuole fruire di questi servizi in una modalità diversa da quella prevista dall'oggetto ~~~~ server

↳ [componenti simili con INTERFACCE DIVERSE]: un oggetto client vuole fruire dei servizi offerti da uno tra più oggetti server che offrono servizi simili - e questi oggetti server hanno interfacce diverse tra loro

Analisi e progettazione del software

20/5/13
XXVI lezione

- inizializzazione pigra -

```
Public class ServicesFactory {
private static ServicesFactory instance = new ServicesFactory();
public static ServicesFactory getInstance() {
    return instance;
} }
```

oss: se è un singleton concettuale molto probabilmente non è un SINGLETON SOFTWARE nè implementazione

23/5/13
XXVII

P. ADAPTER → effettuare l'adattamento verso tutti gli elementi che hanno un'interfaccia compatibile fa

P. FACTORY → ha lo scopo di creare oggetti la cui creazione risulta essere complessa

P. SINGLETON → ha lo scopo di fornire un p.to di accesso globale alla factory per creare gli adattatori per l'accesso ai servizi

→ Come deve essere descritta la soluzione di un problema di progettazione?

elementi fondamentali progettazione
- STRUTTURA STATICA DELLA SOLUZIONE: ovvero che classi mi servono che associazioni mi servono
- CREAZIONE DEGLI OGGETTI:
- USO DEGLI OGGETTI:

→ Quando viene creato l'adattatore? nel caso d'uso di avviamento

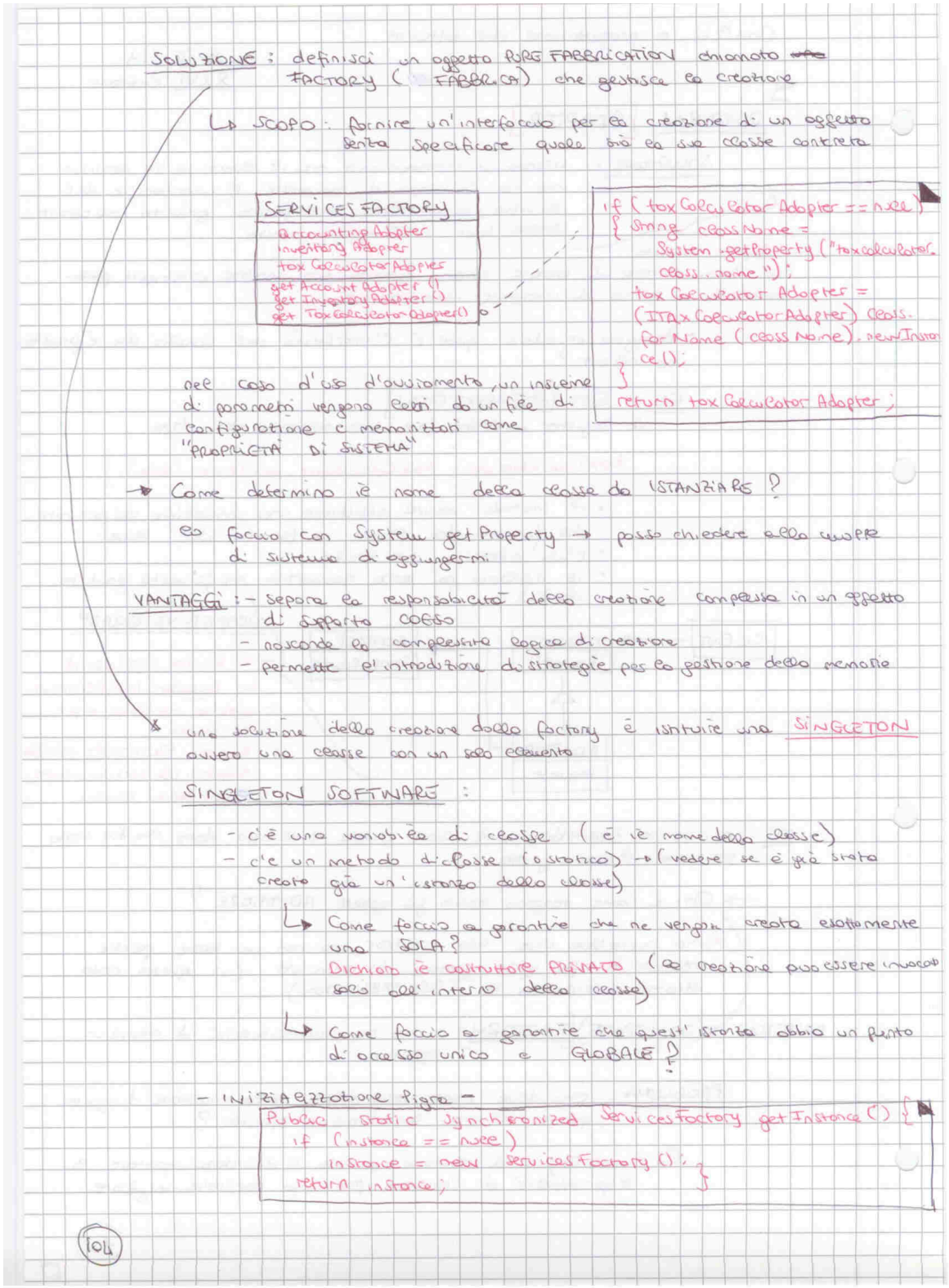
PATTERN [STRATEGY]

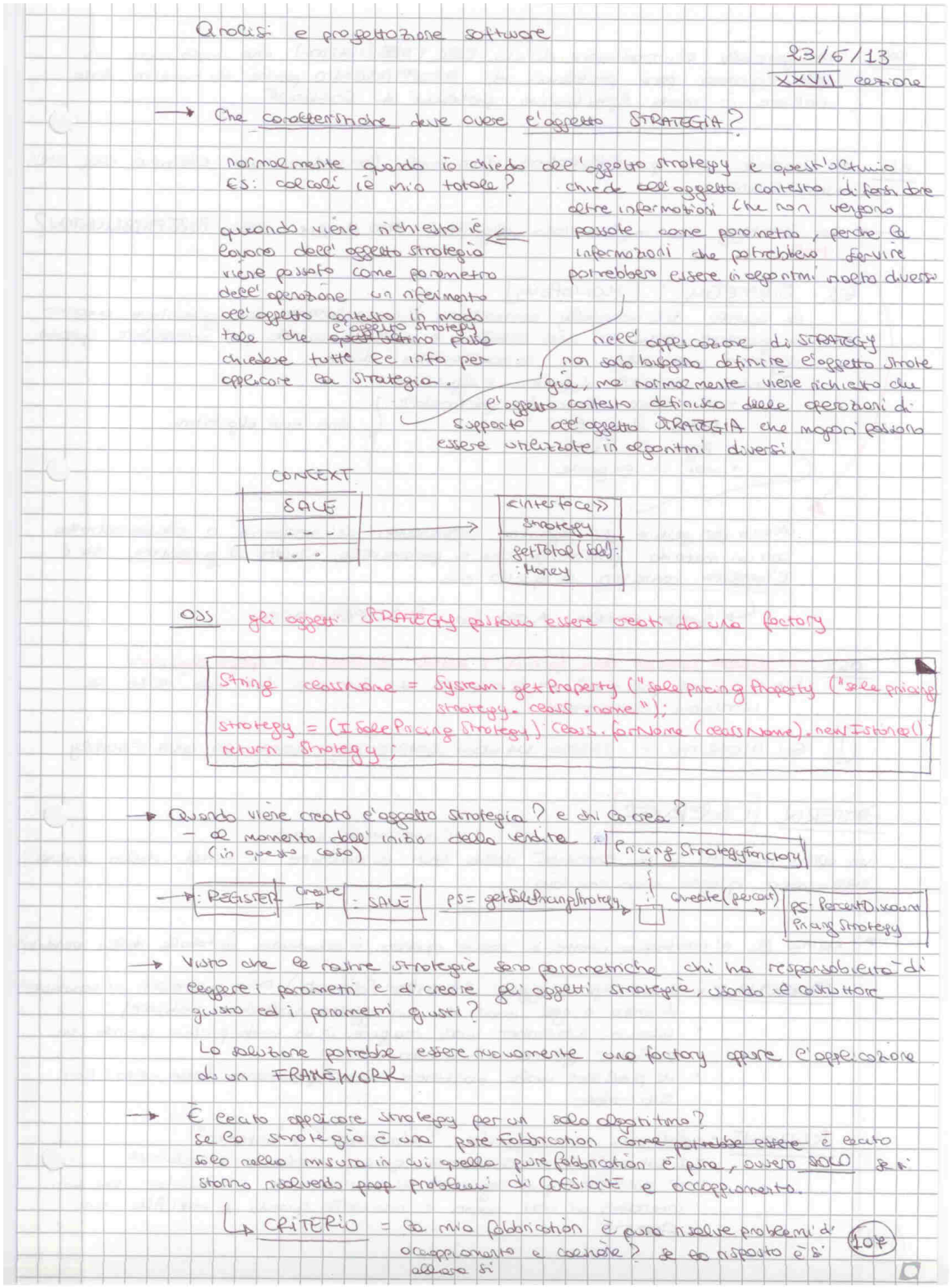
PROBLEMI: (almeno 2 algoritmi)
- Come progettare per gestire un insieme di algoritmi o politiche variabili ma correlate/i?
- Come progettare per consentire di modificare questi algoritmi o politiche?

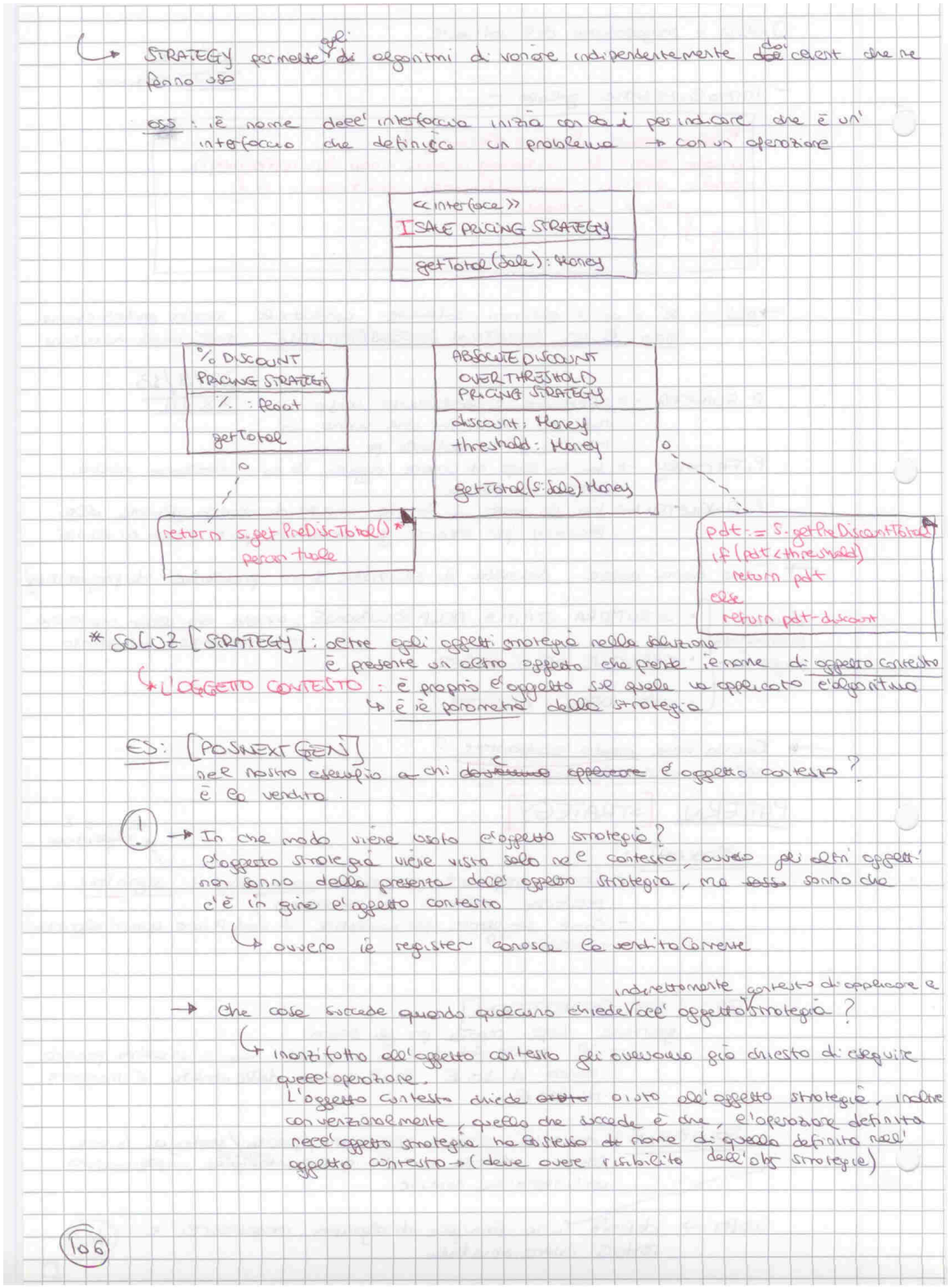
ES: [DIST_POSNEXTGEN]
gestione delle regole per gli sconti
- se sconto dal 10% in un periodo ma in un altro periodo sconto di 10 € se il totale della vendita è maggiore di 200 €

SOLUZIONE: definisci ciascun algoritmo/politica/strategia in una classe diversa e tutte queste classi implementano un'interfaccia comune

SCOPO → definire una famiglia di algoritmi, incapsularli e renderli intercambiabili


SOLUZIONE : definisci un oggetto PURE FABRICATION chiamato FACTORY (FABBRICA) che gestisca la creazione
SCOPO : fornire un'interfaccia per la creazione di un oggetto senza specificare quale sia la sua classe concreta
SERVICES FACTORY
AccountingAdapter
InventoryAdapter
TaxCalculatorAdapter
getAccountAdapter()
getInventoryAdapter()
getTaxCalculatorAdapter()
if (taxCalculatorAdapter == null)
{ String className =
System.getProperty("taxcalculator.
class.name");
taxCalculatorAdapter =
(ITaxCalculatorAdapter) Class.
forName(className).newInstan
ce();
}
return taxCalculatorAdapter;
nel caso d'uso d'avviamento, un insieme di parametri vengono letti da un file di configurazione o memorizzati come "PROPRIETÀ DI SISTEMA"
Come determino il nome della classe da ISTANZIARE ?
la faccio con System.getProperty → passo chiedere alla JVM di sistema di aggiungermi
VANTAGGI : - separa la responsabilità della creazione complessa in un oggetto di supporto COESO
- nasconde la complessità logica di creazione
- permette l'introduzione di strategie per la gestione della memoria
una soluzione della creazione della factory è istituire una SINGLETON ovvero una classe con un solo elemento
SINGLETON SOFTWARE :
- c'è una variabile di classe (il nome della classe)
- c'è un metodo di classe (statico) → (vedere se è già stata creata già un'istanza della classe)
Come faccio a garantire che ne venga creata esattamente una SOLA ?
Dichiaro il costruttore PRIVATO (la creazione può essere invocata solo all'interno della classe)
Come faccio a garantire che quest'istanza abbia un punto di accesso unico e GLOBALE ?
- INIZIALIZZAZIONE PIGRA -
Public static synchronized ServicesFactory getInstance() {
if (instance == null)
instance = new ServicesFactory();
return instance;
}
104


Analisi e progettazione software
23/5/13
XXVII sessione
→ Che caratteristiche deve avere l'oggetto STRATEGIA?
normalmente quando io chiedo es: calcola il mio totale?
quando viene richiesto il lavoro dell'oggetto strategia viene passato come parametro dell'operazione un riferimento dell'oggetto contesto in modo tale che l'oggetto strategy possa chiedere tutte le info per applicare la strategia.
dell'oggetto strategy e quest'ultimo chiede all'oggetto contesto di fornire altre informazioni che non vengono passate come parametro, perché le informazioni che potrebbero servire potrebbero essere in algoritmi molto diversi
nell'applicazione di STRATEGY non solo bisogna definire l'oggetto strategia, ma normalmente viene richiesto che l'oggetto contesto definisca delle operazioni di supporto dell'oggetto STRATEGIA che magari possono essere utilizzate in algoritmi diversi.
CONTEXT
SALE
<<interface>>
Strategy
getTotal(sale): Money
OSS gli oggetti STRATEGY possono essere creati da una factory
String className = System.getProperty("sale.pricing.Strategy", "sale.pricing.Strategy.class.name");
strategy = (ISalePricingStrategy) Class.forName(className).newInstance();
return Strategy;
→ Quando viene creato l'oggetto strategia? e chi lo crea?
– al momento dell'inizio della vendita (in questo caso)
PricingStrategyFactory
: REGISTER
create
: SALE
ps = getSalePricingStrategy
create(percent)
PS: PercentDiscount PricingStrategy
→ Visto che le nostre strategie sono parametriche chi ha responsabilità di leggere i parametri e di creare gli oggetti strategie, usando il costruttore giusto ed i parametri giusti?
Lo soluzione potrebbe essere nuovamente una factory oppure l'applicazione di un FRAMEWORK
→ È giusto applicare strategy per un solo algoritmo?
se la strategia è una pure fabrication come potrebbe essere è giusto solo nella misura in cui quella pure fabrication è pura, ovvero SOLO si sta occupando di risolvere problemi di coesione e accoppiamento.
↳ CRITERIO = la mia fabbrication è pura risolve problemi di accoppiamento e coesione? se la risposta è sì allora sì
10 p


STRATEGY permette gli algoritmi di variare indipendentemente dai client che ne fanno uso
ESS: il nome dell'interfaccia inizia con la I per indicare che è un'interfaccia che definisce un problema → con un operazione
<<interface>>
ISALE PRICING STRATEGY
getTotal(Sale): Money
% DISCOUNT PRICING STRATEGY
%: float
getTotal
ABSOLUTE DISCOUNT OVER THRESHOLD PRICING STRATEGY
discount: Money
threshold: Money
getTotal(s:Sale) Money
return s.getPreDiscTotal() * percentuale
pdt := S.getPreDiscountTotal()
if (pdt < threshold)
return pdt
else
return pdt-discount
* SOLUZ [STRATEGY]: oltre agli oggetti strategia nella soluzione è presente un altro oggetto che porta il nome di: oggetto contesto
L'OGGETTO CONTESTO: è proprio l'oggetto sul quale va applicato l'algoritmo
↳ è il parametro della strategia
ES: [POSNEXTGEN]
nel nostro esempio a chi dobbiamo applicare è oggetto contesto? è la vendita
In che modo viene usato l'oggetto strategia? L'oggetto strategia viene visto solo nel contesto, ovvero gli altri oggetti non sanno della presenza dell'oggetto strategia, ma sanno che c'è in giro l'oggetto contesto
↳ ovvero il register conosce la venditaCorrente
Che cose succede quando qualcuno chiede l'oggetto strategia? indirettamente contesto di applicare
↳ innanzitutto all'oggetto contesto gli viene richiesto di eseguire quell'operazione. L'oggetto contesto chiede aiuto all'oggetto strategia, infatti convenzionalmente quello che succede è che, l'operazione definita nell'oggetto strategia ha lo stesso nome di quella definita nell'oggetto contesto → (deve avere richiesto dell'obj strategia)
106

Analisi e progettazione software

23/5/13
XXVII Lezione

① bisogna oggettificare gli eventi → in buona parte viene fatto con le interfacce.

SCOPO → definire una dipendenza uno a molti fra oggetti in modo tale che se un oggetto cambia il suo stato, tutti gli oggetti dipendenti da questo siano notificati e aggiornati automaticamente

– DESCRIVERE OBSERVER –: significa capire ① la struttura statica ② capire come vengono creati gli oggetti e fatti i collegamenti e ③ capire come avengono le interazioni

→ L'INTERFACCIA LISTENER:

dichiara un metodo:
- che deve essere implementato da ogni Subscriber
- che verrà invocato ogni volta che si verifica un evento di interesse

<< interface >>
PropertyListner
onPropertyEvent(source, name, value)

ES: ciascuna vendita conosce un insieme dei suoi ascoltatori; e l'editor deve conoscere i suoi abbonati

→ Che succede quando si verifica un evento?

ES: quando cambia il totale di una vendita, non solo si fa il totale = nuovoTotale, ma viene invocato il metodo PublishPropertyEvent(name, value) → itera su tutti gli elementi della collezione e invoca su ciascun abbonato onPropertyEvent

→ Come vengono aggiunti abbonati alla collezione di ascoltatori creata dalla vendita?

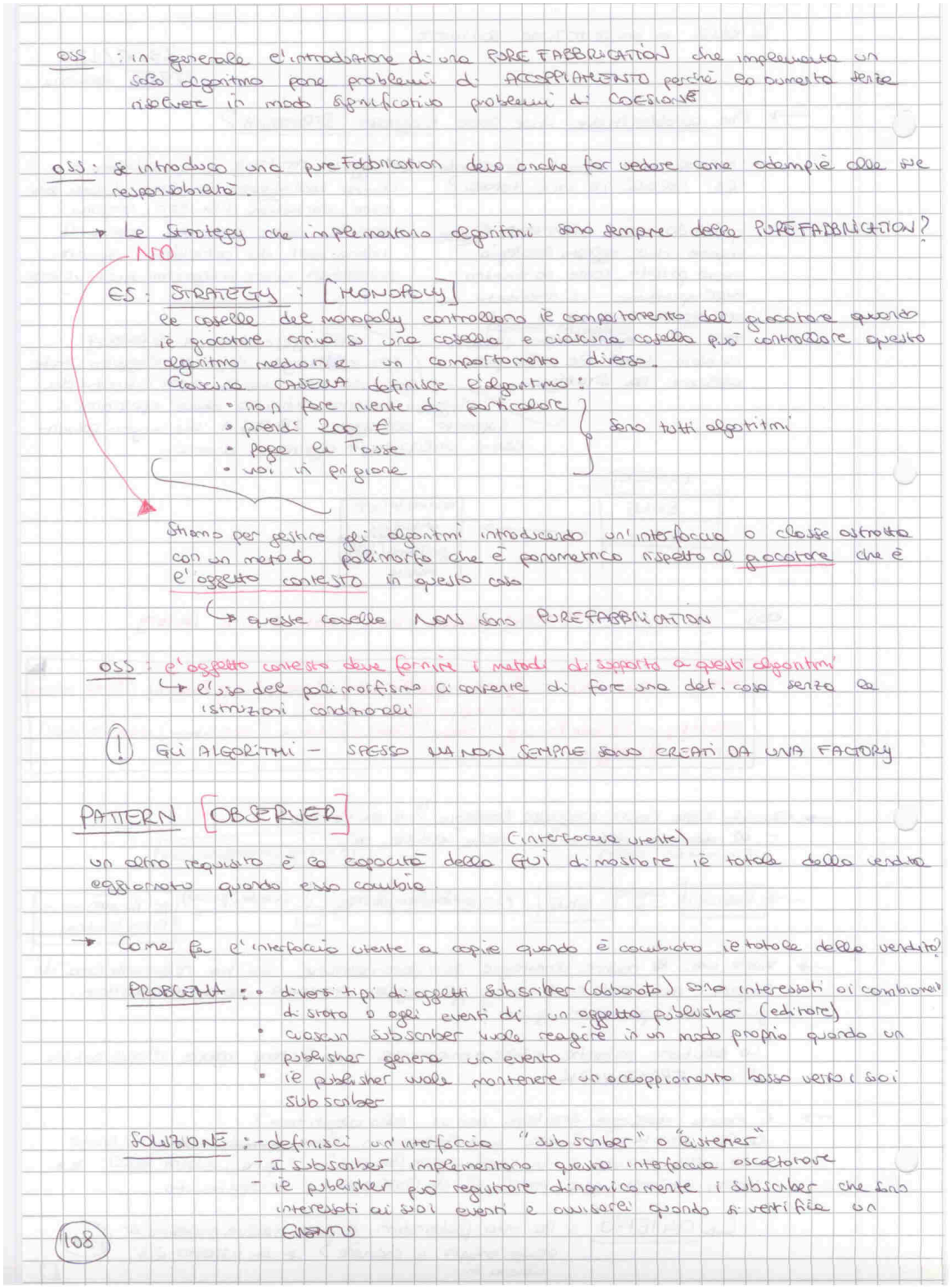
↳ la vendita deve quindi fornire un metodo che richiama addPropertyListner(PropertyListner)

→ Come fa l'interfaccia utente ad essere notificato di questi eventi? ↳ è necessario che ci sia l'abbonamento

OSS: MVC è pesantemente basato su observer

Ci sono diversi tipi di oggetti editore, ci stanno diversi tipi di oggetto abbonati, ci sono diversi tipi di eventi.
Alcuni abbonati sono interessati ad uno o più tipi di eventi, per ricevere notifiche di questi eventi si devono registrare.
Quando sono registrati e questi eventi si verificano allora vengono notificati.

ES: FACEBOOK è basato fortemente su questo



OSS : in generale è introduttore di una PURE FABRICATION che implementa un suo algoritmo pone problemi di ACCOPPIAMENTO perché ha numerosi tipi da invocare in modo significativo problemi di COESIONE

OSS: se introduco una pureFabrication devo anche far vedere come adempie alle sue responsabilità.

→ Le Strategy che implementano algoritmi sono sempre delle PUREFABRICATION?
NO

ES: STRATEGY : [MONOPOLY]
le caselle del monopoly controllano il comportamento del giocatore quando il giocatore arriva su una casella e ciascuna casella può controllare questo algoritmo mediante un comportamento diverso.
Ciascuna CASELLA definisce l'algoritmo:
- non fare niente di particolare
- prendi 200 €
- paga la Tasse
- vai in prigione

} sono tutti algoritmi

Stiamo per gestire gli algoritmi introducendo un'interfaccia o classe astratta con un metodo polimorfo che è parametrico rispetto al giocatore che è l'oggetto contesto in questo caso
↳ queste caselle NON sono PUREFABRICATION

OSS: l'oggetto contesto deve fornire i metodi di supporto a questi algoritmi
↳ l'uso del polimorfismo ci consente di fare una determinata cosa senza le istruzioni condizionali

(!) GLI ALGORITMI – SPESSO MA NON SEMPRE SONO CREATI DA UNA FACTORY

PATTERN [OBSERVER]

un altro requisito è la capacità della GUI di mostrare (interfaccia utente) la totalità della vendita aggiornata quando esso cambia

→ Come fa l'interfaccia utente a capire quando è cambiato il totale delle vendite?

PROBLEMA :
- diversi tipi di oggetti subscriber (abbonati) sono interessati ai cambiamenti di stato o agli eventi di un oggetto publisher (editore)
- ciascun subscriber vuole reagire in un modo proprio quando un publisher genera un evento
- il publisher vuole mantenere un accoppiamento basso verso i suoi subscriber

SOLUZIONE :
- definisci un interfaccia "subscriber" o "listener"
- I subscriber implementano questa interfaccia ascoltatore
- il publisher può registrare dinamicamente i subscriber che sono interessati ai suoi eventi e avvisarli quando si verifica un EVENTO

Analisi e progettazione software

27/5/13

XXVIII lezione

PROXY esiste in tante varianti:

- CACHE PROXY: diversi client locali possono condividere i risultati da componenti remoti
  Assegnare la responsabilità ad un intermediario che gestisca meglio questo servizio
- VIRTUAL PROXY: caricamento pigro dei dati; io quando ad esempio vado a cercare dalla base di dati una persona dovrei andare a trovare tutte le "macchine" di quella persona, però caricamento PIGRO non voglio caricare i dati delle sue automobile
  → quello che faccio è di creare un oggetto VIRTUAL PROXY per la macchina che ha la stessa interfaccia della macchina, ma in realtà quest'ultimo contiene una sola chiave della macchina.
- REMOTE PROXY: vuol dire che il servizio è distribuito su un altro calcolatore, quindi c'è un oggetto che deve chiedere ad un altro oggetto di eseguire un altra operazione
- PROTECTION PROXY: c'è un servizio e dei client che vogliono usare il servizio però non tutti i client possono utilizzare il servizio c'è bisogno di avere delle credenziali. Chi controlla le credenziali è l'intermediario perché il client potrebbe barare, mentre il servente non ha la questa responsabilità (deve solamente erogare il servizio)

in sostanza il problema affrontato da proxy è quello di sostenere una certa qualità (NO aspetto funzionale)
→ viene assegnato sempre assegnato ad oggetti diversi da quelli di dominio

PATTERN [FACADE]

deve essere totalmente indipendente dalla soluzione tecnologica

PROBLEMA: - è richiesta un'interfaccia comune e unificata per un insieme disparato di implementazioni o interfacce
- può verificarsi un accoppiamento indesiderato a molti oggetti nel sottosistema, oppure l'implementazione del sottosistema può cambiare

SOLUZIONE: - definisci un punto di contatto singolo con il sotto-sistema ovvero una facade
- questo oggetto facade presenta un'interfaccia singola e unificata ed è responsabile della collaborazione tra i componenti del sottosistema

OSS: quando ho un interfaccia devo creare un oggetto

↳ SCOPO: fornire un'interfaccia unificata per un insieme di interfacce presenti in un sottosistema
- FACADE definisce un'interfaccia di livello più alto che rende il sottosistema più semplice da utilizzare

DISCUSSIONE OBSERVER : tutto questo giochetto si regge perché qualcuno dice all'interfaccia utente, quando è cambiata la vendita.
Perché se inizia una nuova vendita e l'interfaccia utente non lo sa? ~~ie totale della vendita non viene mostrato~~

l'interfaccia utente si deve registrare agli eventi della vendita corrente, quindi deve sapere qual'è la vendita corrente e deve sapere quando farlo.
Affinché questo sia possibile, l'interfaccia utente è interessata ad eventi del tipo è iniziata una nuova vendita

→ chi è in grado di notificare eventi di questo tipo?

↳ è ie controller (register). Quindi la sua è interessata anche agli eventi generati dal controller e deve abbonarsi presso la vendita

OSS : observer non si applica solo all'interfaccia utente.

PATTERN [PROXY]

27/5/13
XXVIII lezione

proxy è una forma di indirezione → questo è simile ad adapter anche se in realtà è tutta un altra cosa.

- Ie proxy in generale è un intermediario che ha lo scopo di supportare una certa qualità che ha ie servizio. (voglio migliorare ie servizio)

es : Gruppo di condivisa

PROBLEMA : c'è un client ed un server e l'accesso del servizio da parte del client ~~e l'accesso del client~~ al server è possibile ma si potrebbe migliorare secondo:

- sicurezza
- efficienza
- prestazioni

SOLUZIONE : introdurre un intermediario in modo tale che quando ie client vuole accedere al server, in verità accede al proxy, e poi è ie proxy che accede al server al posto del client.
Un aspetto importante è che quando ie proxy accede al server, prima dell'accesso al server e dopo l'accesso al server, ie servizio può eseguire delle PRE-ELABORAZIONI e delle POST-ELABORAZIONI.

PROXY VS ADAPTER : proxy è diverso da adapter perché qui l'interfaccia del proxy è proprio quella del servente, mentre in adapter non è così a causa di un'incompatibilità di interfaccia

↳ ie fatto che l'interfaccia sia la stessa è importante, perché quello che spesso succede è che ie client pensa di star usando direttamente ie servizio ma in realtà lo sta usando indirettamente tramite ie proxy

Analisi e progettazione software

27/5/13
XXVIII Lezione

MVC: assume la presenza di più viste e di più controller associati al sistema tipicamente coppie viste controller e c'è un solo modello
se ci sono più viste per accedere agli stessi stati bisogna utilizzare OBSERVER

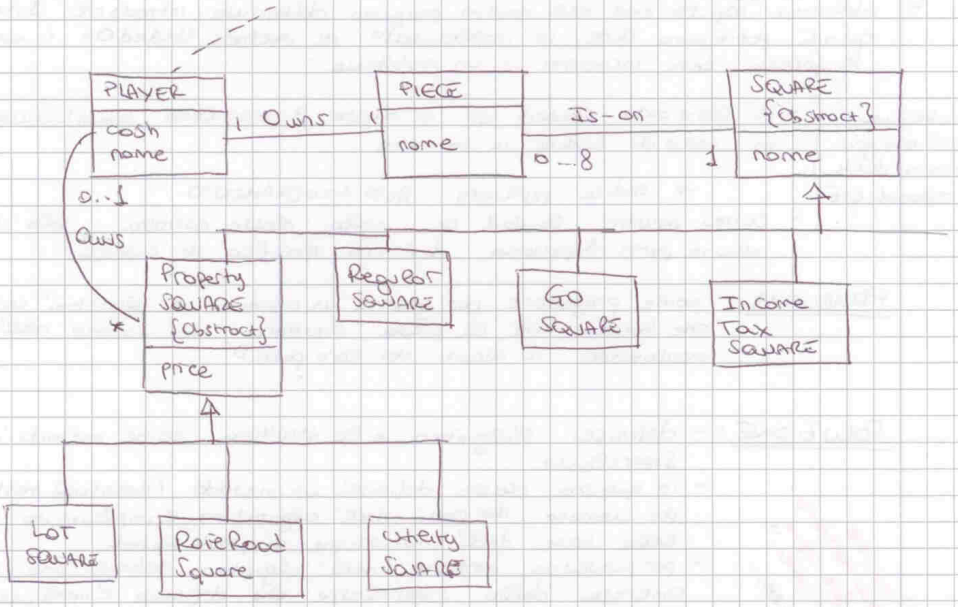
OSS: chi ha la responsabilità di creare dei PROXY?
↳ delle FACTORY visto che PROXY è una pure fabbrication

PROGETTAZIONE & PATTERN GOF (ULTERIORI)

TERZA ITERAZIONE: [MONOPOLY]

- 3 tipi di caselle proprietà ① Caselle terreno → si paga un affitto che dipende dal terreno
  ② Caselle stazione: si paga un affitto che dipende dal numero di stazioni possedute dal suo proprietario
  ③ Caselle società: affitto prop. dal tiro dei dadi

Come introduco queste caselle nel MODELLO DI DOMINIO?



OSS: terreni società e stazioni devono essere rappresentati da 3 classi diverse

① ogni volta che si introduce una superclasse è bene che sia ASTRATTA

FACADE Vs ADAPTER : sono cose diverse perché inanzitutto di adapter ce ne sono tanti ~~mentre invece~~ mentre invece quando si fa facade è implementazione è una sola.
Inoltre adapter tipicamente è legato all'accesso di cose pre-esistenti mentre facade spesso è legato a cose che non si sa come devono essere fatte oppure non si vuole che si sappia come devono essere fatte

COLLABORAZIONE TRA GLI STRATI

all'interno di uno strato tutti possono collaborare con tutti.

→ Come sono possibili le collaborazioni tra strati?

↳ Inanzitutto visto che noi vediamo il mondo dal punto di vista della logica applicativa. Dobbiamo distinguere tra le collaborazioni verso il basso e le collaborazioni verso l'alto

- un sottosistema è un insieme complesso di elementi che si compone di uno o più package

↳ il PATTERN di accesso più comune ai sottosistemi è una FACADE
LE COMUNICAZIONI VERSO IL BASSO SONO GENERALMENTE ORGANIZZATE E ACCEDUTE TRAMITE UNA FACADE

- sia lo strato di dominio che lo strato della "persistenza" o altro usa una FACADE per accedere ai servizi di quel sottosistema

↳ COLLABORAZIONI VERSO L'ALTO : di quale natura potrebbero essere?

ES: il totale della vendita che cambia e io voglio che il totale della vendita venga aggiornato

↳ NOTIFICHE : comunicazioni dal basso verso l'alto, poiché quest'ultima è una parola associata ad evento.
Evento significa che si è verificato qualche cosa di rilevante quindi le comunicazioni dal basso verso l'alto sono motivate dal fatto che si è verificato un evento e devo notificare qualcuno del fatto che si è verificato questo evento.

→ COME SONO POSSIBILI le comunicazioni verso l'alto?

viene fatto attraverso una FACADE ← 1) PULL → realizzato tramite polling, periodicamente vado a vedere cose è successo (su richiesta)

2) PUSH → quando si verifica un evento (dal basso verso l'alto) questo viene immediatamente notificato

possono essere basate su OBSERVER, ovvero chi ha bisogno di certi eventi si deve ~~notificare~~ abbonare presso la sorgente degli eventi, e questo garantisce

[oppure utilizzare una facade (solo in alcuni casi) quando ad esempio l'elemento è uno solo che ogni volta che si verifica un evento viene immediatamente informato]

Analisi e progettazione del software
Anno Accademico 2012-2013
Acme Payroll – Requisiti

AcmePayoll (nel seguito chiamato semplicemente "sistema") è il sistema software usato dal ***Laboratorio Acme*** (*it.wikipedia.org/wiki/Laboratorio_ACME*, nel seguito, "Acme") per il calcolo delle buste paga dei suoi impiegati.
Ogni volta che un impiegato della Acme va a lavorare, deve "timbrare" il proprio cartellino. Il sistema deve registrare opportunamente le presenze dei propri impiegati, memorizzando per ciascuna presenza la data, l'ora di entrata e l'ora di uscita.
Come noto, la Acme produce marchingegni. Ciascuna tipologia di marchingegno ha un nome (ad esempio, *Pianoforte esplosivo*) e un prezzo di vendita (ad esempio, *5000 euro*). Ogni volta che un impiegato (o un gruppo di impiegati) della Acme completa la produzione di un marchingegno, il sistema deve registrare opportunamente il marchingegno creato, memorizzando anche la data di completamento e l'impiegato (o il gruppo di impiegati) che lo ha realizzato.
Per semplicità, si supponga che il sistema memorizzi solo le presenze degli impiegati dell'ultimo mese, e solo i dati sui marchingegni completati nell'ultimo mese.
Alla fine del mese, il sistema deve calcolare quanto deve essere pagato ciascuno degli impiegati. Gli impiegati vengono pagati in base alle loro presenze presso la Acme e a quanti e quali marchingegni hanno costruito. Per la precisione, ciascun impiegato può accordarsi con la Acme per il pagamento della propria attività in base ad alcuni tipi di contratti predefiniti. I tipi di contratti di interesse in questa iterazione sono:

- **Pagamento forfettario**: In un mese, l'impiegato viene pagato un certo importo fisso, indipendentemente dalle sue presenze e dal numero di marchingegni che ha costruito in quel mese. L'importo fisso può variare da impiegato ad impiegato, ed è scritto nel contratto dell'impiegato. Ad esempio, 2000 euro al mese.
- **Pagamento a giorni di presenza**: In un mese, l'impiegato viene pagato in base al numero di giorni in cui è stato effettivamente presente in quel mese, indipendentemente dal numero di marchingegni che ha costruito e dal numero di ore di presenza effettiva. L'importo pagato per ciascun giorno di presenza può variare da impiegato ad impiegato, ed è scritto nel contratto dell'impiegato. Ad esempio, 400 euro a giorno di presenza.
- **Pagamento a ore di presenza**: In un mese, l'impiegato viene pagato in base al numero di ore in cui è stato effettivamente presente in quel mese, indipendentemente dal numero di marchingegni che ha costruito e dal numero di giornate di presenza. L'importo pagato per ciascuna ora di presenza può variare da impiegato ad impiegato, ed è scritto nel contratto dell'impiegato. Ad esempio, 30 euro a ora di presenza.
- **Pagamento per marchingegni**: In un mese, l'impiegato viene pagato in base ai marchingegni che ha costruito in quel mese, indipendentemente dalle sue presenze in quel mese. L'importo pagato per ciascun marchingegno costruito è dato dal prodotto del prezzo di vendita del marchingegno, diviso il numero di impiegati che ha realizzato il marchingegno, e moltiplicato per una percentuale, che può variare da impiegato ad impiegato, ed è scritta nel contratto dell'impiegato. Ad esempio, il 10% della quota per impiegato del marchingegno. (Ad esempio, se un impiegato ha realizzato un solo *Pianoforte esplosivo*, il cui prezzo è 5000 euro, e questo è stato fatto da un gruppo di cinque impiegati, allora la quota per impiegato è 1000 euro. Se l'impiegato viene pagato con una percentuale del 10%, allora gli spettano 100 euro.)

Ogni impiegato si accorda con la Acme per un contratto mensile (questo può avvenire in qualunque giorno del mese, sia all'inizio che poco prima del pagamento). Il contratto mensile di un impiegato può cambiare di mese in mese. Il sistema deve memorizzare tutti i contratti degli impiegati, sia quelli relativi al mese corrente che a quelli precedenti.

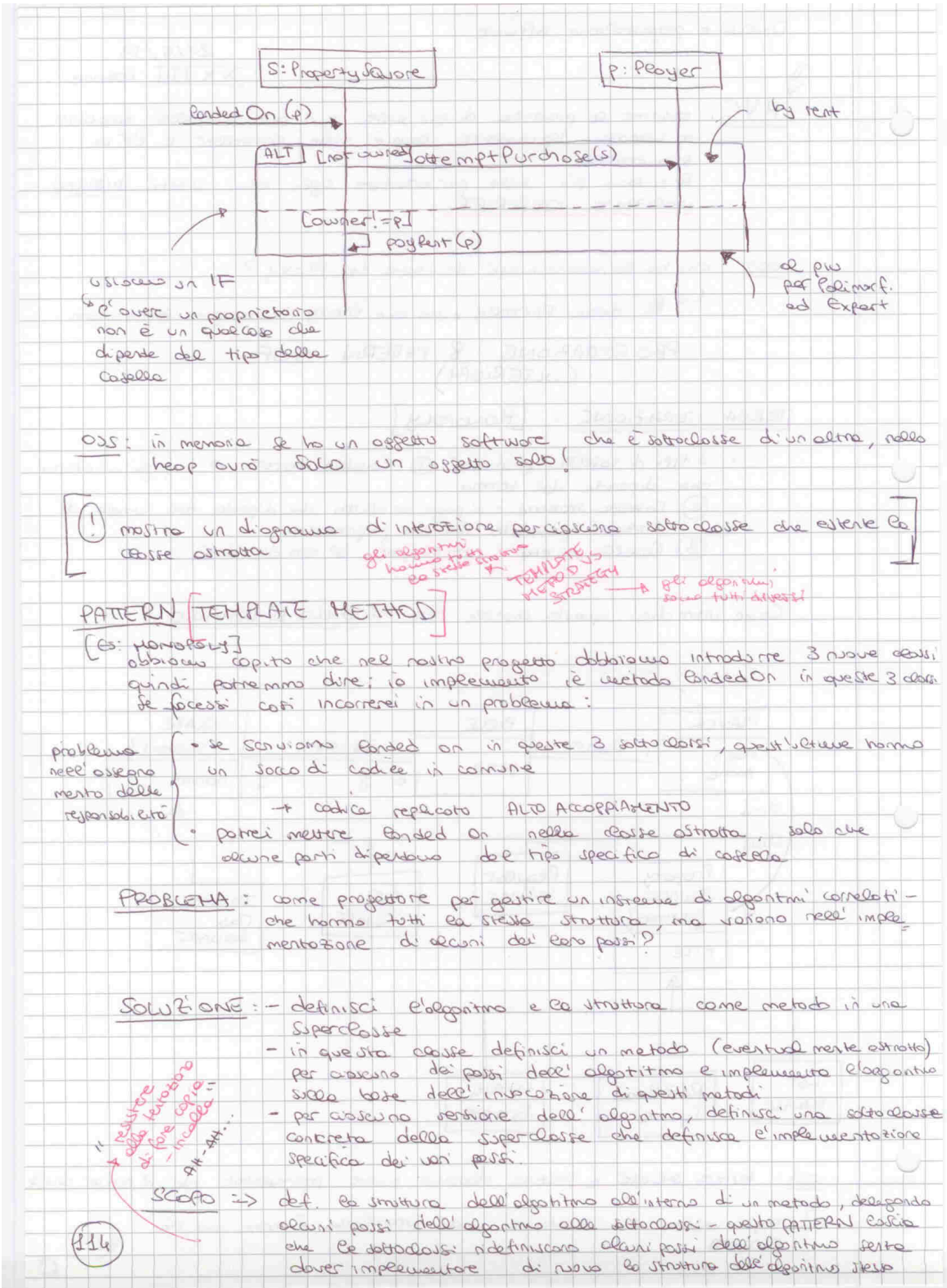
L'uso del sistema in discussione è descritto principalmente dai seguenti casi d'uso, descritti in modo sintetico:

Caso d'uso UC1: Gestione presenza – *Attore primario: un Impiegato.*
L'Impiegato usa il Sistema per registrare la propria entrata alla Acme o la propria uscita dalla Acme. Il Sistema registra (o aggiorna) la presenza.

Caso d'uso UC2: Completamento marchingegno – *Attore primario: un Impiegato.*
L'Impiegato usa il Sistema per registrare il completamento di un marchingegno. L'Impiegato indica il tipo di marchingegno e il gruppo di Impiegati che ha costruito il marchingegno. Il Sistema registra il marchingegno.

Caso d'uso UC3: Contratto impiegato – *Attore primario: l'Ufficio del Personale.*
L'Ufficio del Personale usa il Sistema per registrare, per un certo Impiegato, il contratto da utilizzare per quell'Impiegato in quel mese, con i relativi parametri.

Caso d'uso UC4: Pagamento impiegato – *Attore primario: l'Ufficio del Personale.*
L'Ufficio del Personale usa il Sistema per calcolare, per un certo Impiegato, l'importo che va pagato a quell'Impiegato, per il lavoro svolto nell'ultimo mese.


S: PropertySeller
p: Player
landedOn(p)
ALT
[not owned] attemptPurchase(s)
by rent
[owner!=p]
payRent(p)
blocco in IF
è avere un proprietario non è un qualcosa che dipende dal tipo della cella
al più per Polimorf. ad Expert
OSS: in memoria se ho un oggetto software che è sottoclasse di un altra, nello heap avrò solo un oggetto solo!
mostro un diagramma di interazione per ciascuna sottoclasse che estende la classe astratta
gli algoritmi hanno tutti la stessa struttura
TEMPLATE METHOD vs STRATEGY
gli algoritmi sono tutti diversi!
PATTERN [TEMPLATE METHOD]
[es: MONOPOLY]
abbiamo capito che nel nostro progetto dobbiamo introdurre 3 nuove classi quindi potremmo dire: io implementerò il metodo landedOn in queste 3 classi se facessi così incorrerei in un problema:
problema nell'assegnamento delle responsabilità
• se scriviamo landed on in queste 3 sottoclassi, queste avranno un pezzo di codice in comune
→ codice replicato ALTO ACCOPPIAMENTO
• potrei mettere landed on nella classe astratta, solo che alcune parti di pertanto del tipo specifico di cella
PROBLEMA: come progettare per gestire un insieme di algoritmi correlati - che hanno tutti la stessa struttura, ma variano nell'implementazione di alcuni dei loro passi?
SOLUZIONE: - definisci l'algoritmo e la struttura come metodo in una superclasse
- in questa classe definisci un metodo (eventualmente astratto) per ciascuno dei passi dell'algoritmo e implementa l'algoritmo sulla base dell'invocazione di questi metodi
- per ciascuna versione dell'algoritmo, definisci una sottoclasse concreta della superclasse che definisce l'implementazione specifica dei vari passi
SCOPO => def. la struttura dell'algoritmo all'interno di un metodo, delegando alcuni passi dell'algoritmo alle sottoclassi - questo PATTERN evita che le sottoclassi ridefiniscano alcuni passi dell'algoritmo senza dover implementare di nuovo la struttura dell'algoritmo stesso
114

Analisi e progettazione del software
Anno Accademico 2012-2013
Homework 5

Regole: **SCRIVERE IN MODO LEGGIBILE E NON AMBIGUO**. Scrivere il proprio nome su ciascun foglio utilizzato, in alto a destra. Accanto allo svolgimento di ciascun esercizio o domanda va indicato chiaramente l'esercizio o la domanda a cui lo svolgimento è relativo (ad esempio, Esercizio A1, modello di dominio, oppure Esercizio A2, domanda A2.2). Alle domande in cui è prevista una risposta binaria SI/NO ("è opportuno fare questo?") è necessario scrivere in modo esplicito prima la risposta alla domanda (SI oppure NO) e poi dare una breve motivazione della risposta.

In questo homework si faccia riferimento ai requisiti per il sistema **AcmePayroll**, descritti in un documento separato

Ipotesi di lavoro, valide per tutti gli esercizi di progettazione.

- In tutti gli esercizi che seguono, si faccia l'ipotesi che il sistema in discussione gestisca i propri dati solo in memoria principale. Si supponga anche che durante il caso d'uso di avviamento vengano creati e caricati in memoria tutti gli oggetti le cui informazioni siano già effettivamente disponibili al momento dell'avviamento.
- Per ciascuna operazione di sistema va creato un diagramma di interazione che descrive l'interazione relativa alla trasformazione (cambiamento di stato) provocata dall'operazione di sistema. Per quanto riguarda invece le relative risposte (interrogazioni) eventualmente restituite dal sistema, se non è richiesto esplicitamente allora non bisogna mostrare nei diagrammi di interazione né il calcolo dei dati da restituire né la loro visualizzazione. Tuttavia, per le risposte del sistema, è comunque necessario verificare che i dati da restituire possano essere (facilmente) calcolati sulla base delle navigabilità tra gli oggetti che sono state progettate (vedi anche il punto successivo).
- **Le soluzioni individuate dovranno essere compatibili (in particolare in termini di visibilità, ovvero di navigabilità delle associazioni) con le realizzazioni di tutti i casi d'uso mostrati (UC1-UC4).**
- **Nei diagrammi di interazione, mostrare IN MODO ESPLICITO: tutti i MESSAGGI scambiati tra oggetti, tutte le CREAZIONI di oggetti e tutte le FORMAZIONI e ROTTURE di collegamenti.**
- Nei diagrammi di interazione, motivare le scelte di progetto fatte indicando i pattern GRASP e GoF applicati.
- Nei diagrammi delle classi di progetto, mostrare: (1) per ciascuna classe: il nome della classe, i nomi dei suoi attributi, i nomi delle sue operazioni; e (2) per ciascuna associazione e ciascuna sua estremità navigabile: la freccia di navigabilità, il nome dell'estremità, la molteplicità e, in caso di associazione navigabile a molti, il tipo di collezione scelta.

Esercizio A1 (Modellazione di dominio)
Fare l'analisi a oggetti per il sistema in discussione, come segue:

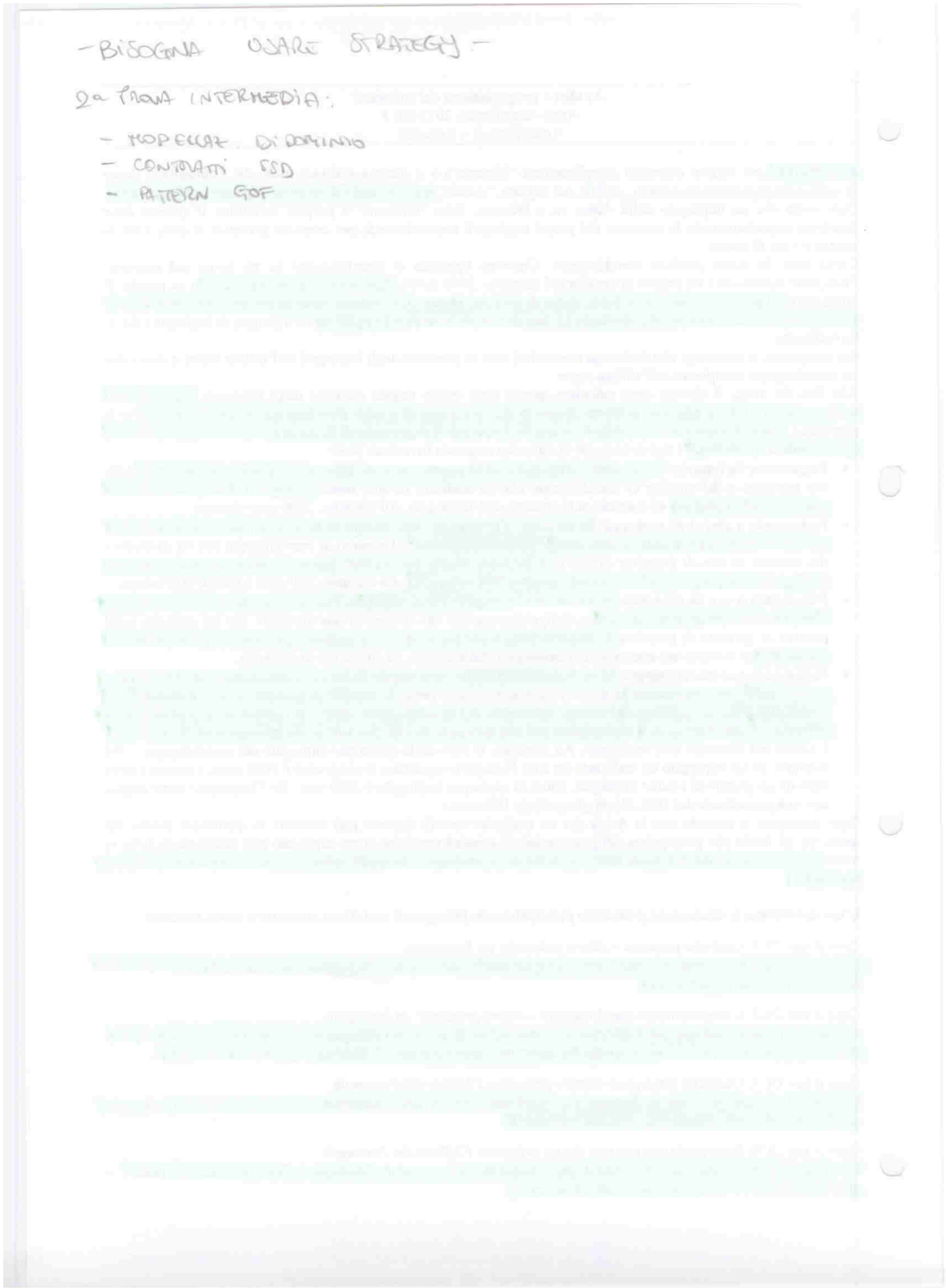
- Mostrare il modello di dominio, relativo a tutti casi d'uso mostrati (UC1-UC4).

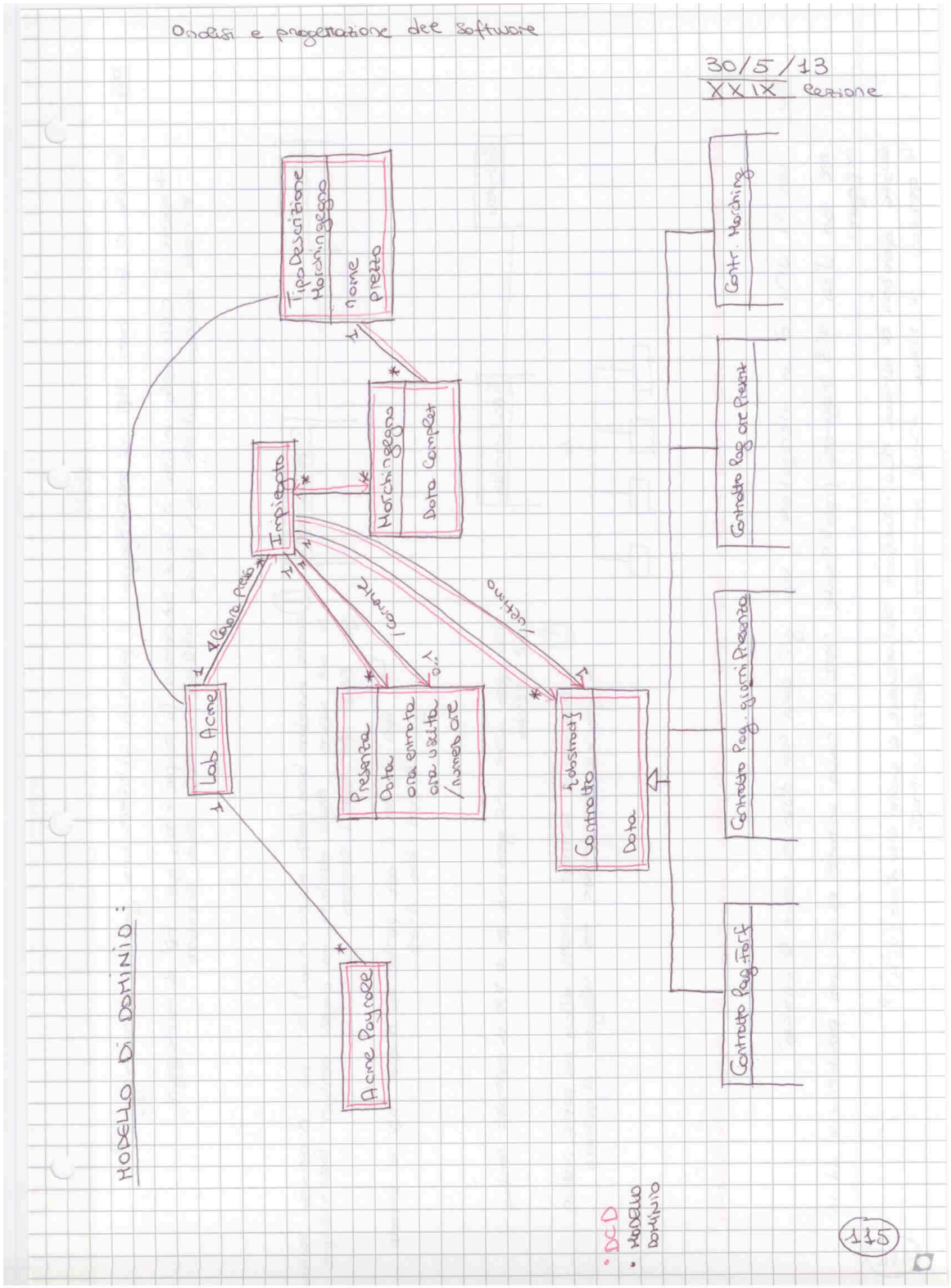
Esercizio A2 (Progettazione)
Considerando per questo esercizio tutti i tipi di contratti per il pagamento degli impiegati, fare la progettazione a oggetti per il sistema in discussione, come segue:

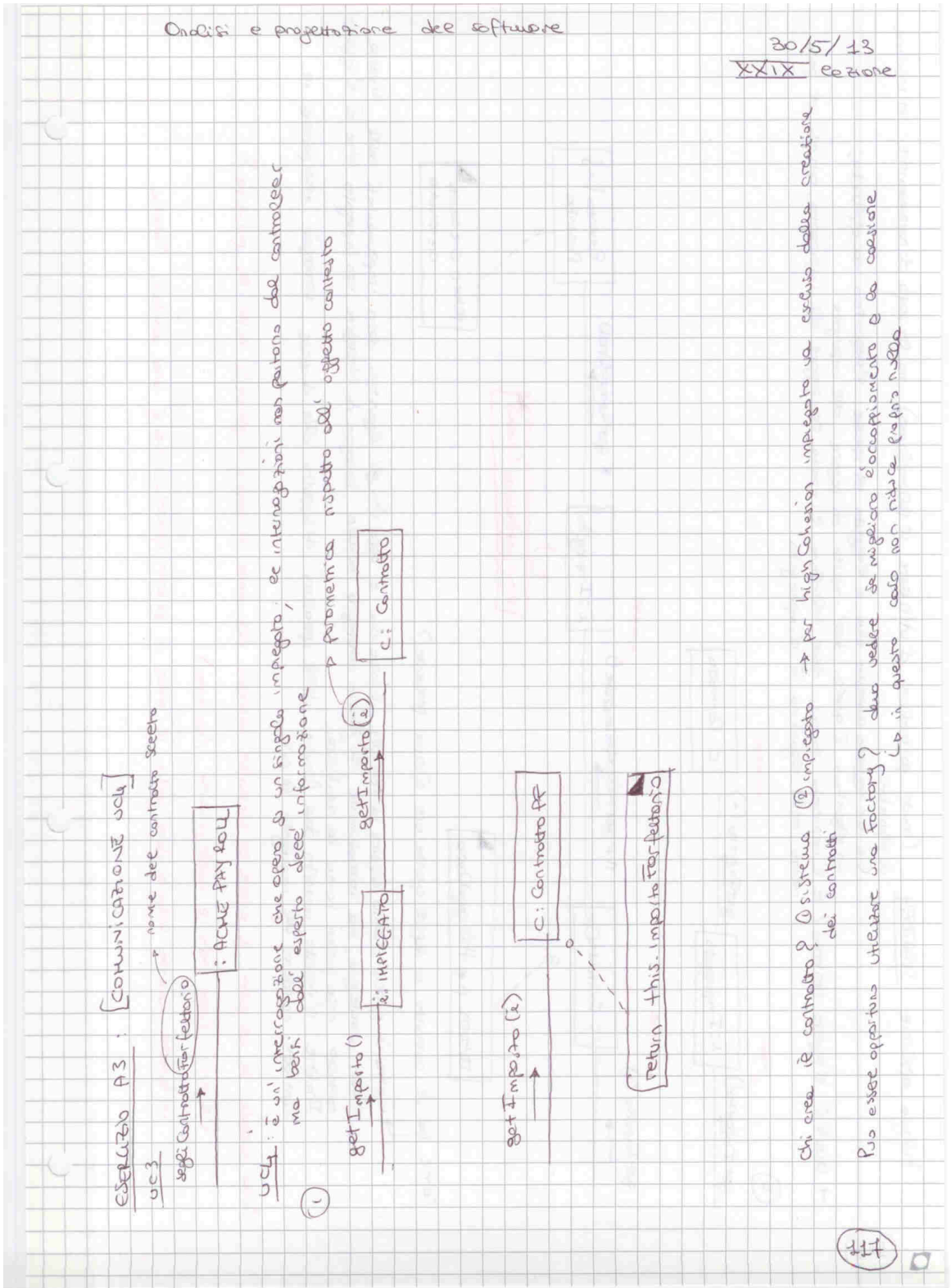
- Mostrare il diagramma delle classi di progetto che si intende realizzare relativamente ai casi d'uso UC1-UC3. In particolare, mostrare le classi e le interfacce che si intendono utilizzare, nonché gli attributi e le associazioni (con le relative molteplicità e navigabilità).

Esercizio A3 (Progettazione)
Considerando per questo esercizio tutti i tipi di contratti per il pagamento degli impiegati, fare la progettazione a oggetti per il sistema in discussione, relativamente al caso d'uso UC4 (*Pagamento impiegato*), come segue:

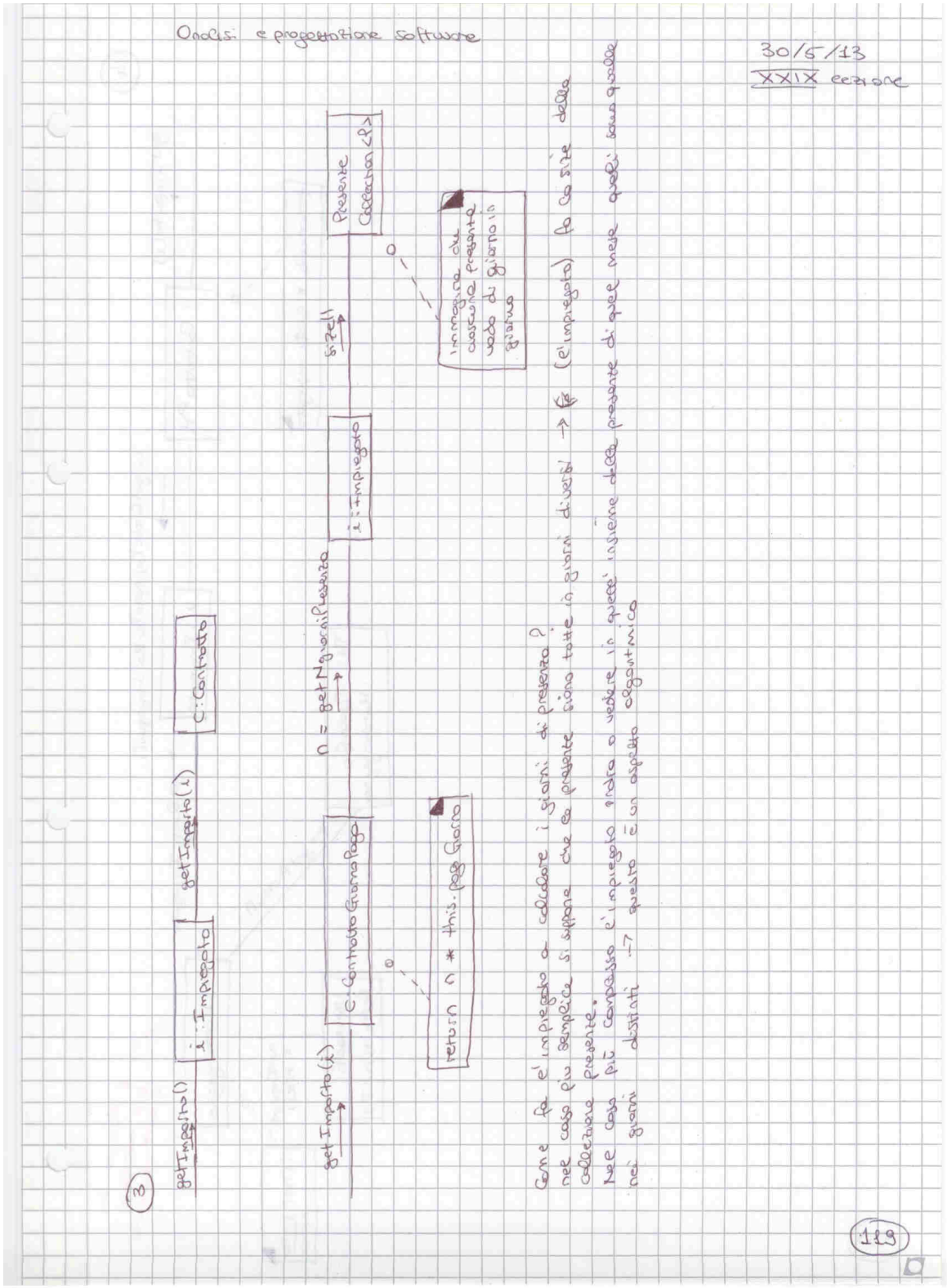
- Mostrare i diagrammi di interazione relativi al calcolo dell'importo da pagare ad un impiegato per il lavoro svolto nell'ultimo mese.

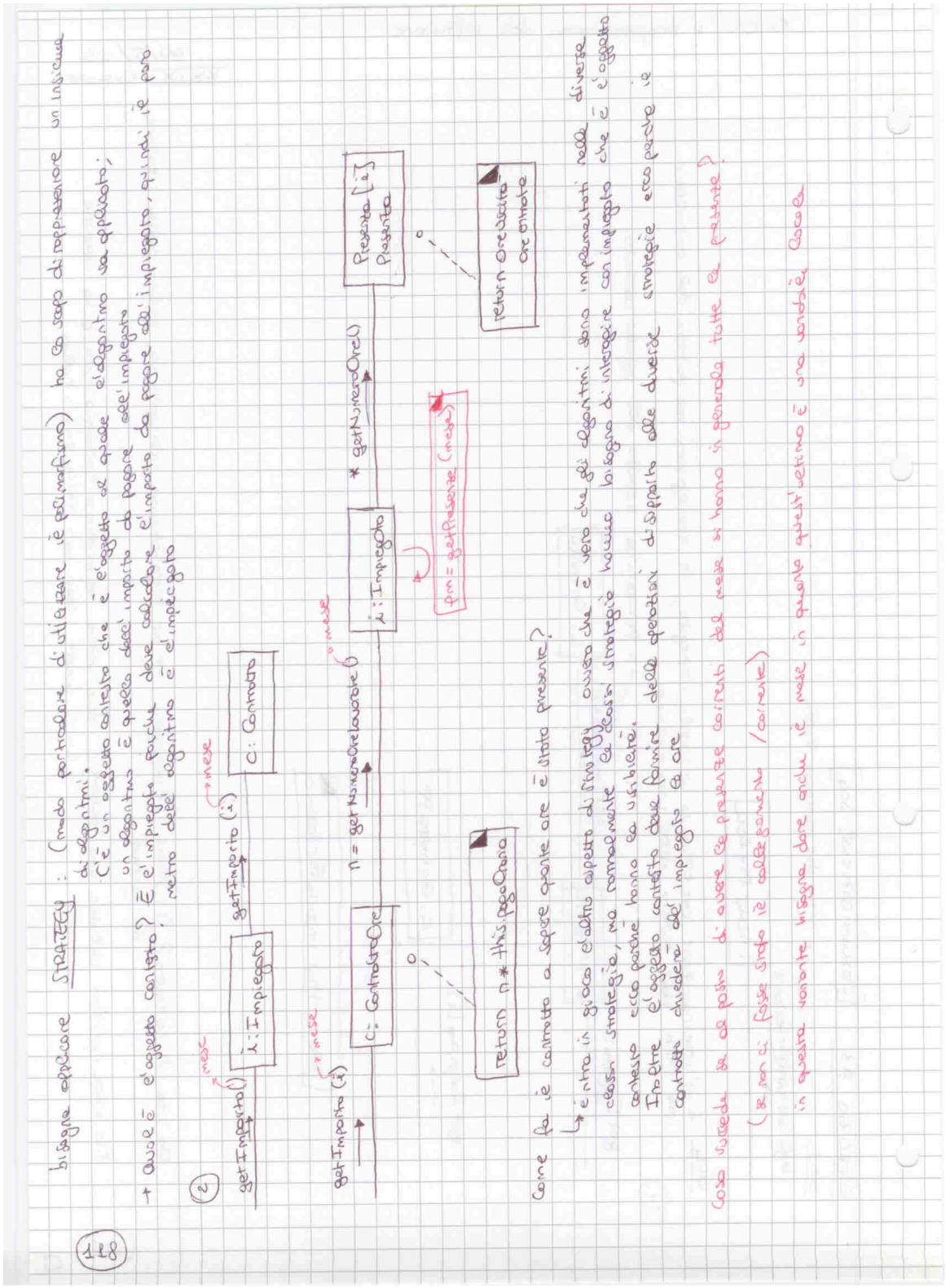

- BISOGNA USARE STRATEGY -
2a PROVA INTERMEDIA:
- MODELLI DI DOMINIO
- CONTRATTI SSD
- PATTERN GOF


Analisi e progettazione del Software
30/5/13
XXIX Lezione
Tipo Descrizione Macchinaggio
nome
prezzo
Impiegato
Macchinaggio
Data Compleat
Lab Acme
Presenza
Data
ora entrata
ora uscita
numero ore
Contratto
Data
Acme Payroll
Contr. Macchinaggio
Contratto Pag. gg Present
Contratto Pag. giorni Presenza
Contratto Pag. forf
MODELLO DI DOMINIO:
DCD
MODELLO DOMINIO
115


Analisi e progettazione del software
30/5/13
XXIX lezione
ESERCIZIO A3 : [COMUNICAZIONE UC4]
UC3
segg Controller(fattorio)
nome del contratto scelto
: ACME PAY POLL
UC4 : è un'interrogazione che opera su un singolo impiegato, ed interagisce con gli altri oggetti del controller
ma basa' dall'esperto delle informazione
(1)
getImpiegato()
i: IMPIEGATO
getImpiegato(i)
c: Contratto
parametrico rispetto all'oggetto contesto
getImpiegato(i)
c: Contratto PF
return this.impiegato.getStipendio()
Chi crea le contratto? Il sistema (2) impiegato → per high Cohesion impiegato va creato dalla creazione dei contratti
Può essere opportuno utilizzare una Factory? dove vedere se migliora l'accoppiamento e la coesione in questo caso non riduce proprio molto
117

Contratto
Tipo Contratto
Contratto
Tipo Contratto


Analisi e progettazione software
30/5/13
XXIX lezione
getImpiegato()
i : Impiegato
getImporto(x)
c : Contratto
getImporto(x)
e : Contratto GiornataPiena
n = getNgiorniPresenza
i : Impiegato
sipeti
Presente Collection<P>
return n * this.pagaGiorno
Immagina che ciascuna presenza vada di giorno in giorno
Come fa e' impiegato a calcolare i giorni di presenza?
nel caso piu semplice si suppone che ce presente fino tutte le giornate diverse → (e' impiegato) fa ce sta dare collezione presente.
Nel caso piu complesso e' impiegato anche o vedere in questo insieme delle presenze di quale mese quali sono giorni distinti → questo e un aspetto algoritmico
3
189


STRATEGY
getImpiegato(i)
getImpiegato(i)
c: Contratto
i: Impiegato
c: Contratto
return c*this.pagaCorpo
i = getNumeroImpiegato()
i: Impiegato
pm = getPesante(mese)
* getNumeroOre()
Pesante [i]
Pesante
return oreSvolte
oreSvolte

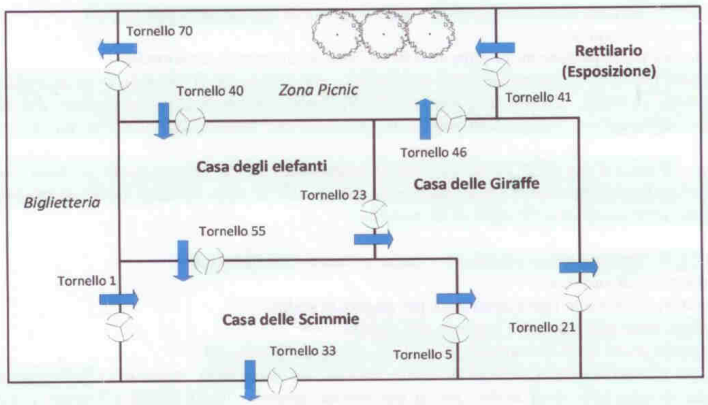
Analisi e progettazione del software
Anno Accademico 2012-2013
Zoo Acme – Requisiti

Lo ***Zoo Acme*** (nel seguito chiamato semplicemente Zoo) è uno zoo "tradizionale" (è composto di diverse aree, in cui vivono numerosi animali) ma offre una modalità innovativa di pagamento delle visite, basato su un sistema di monitoraggio delle presenze dei visitatori nelle diverse aree dello Zoo.

Lo Zoo è composto da tre tipi diversi di aree, come segue:

- Case di animali, in cui vivono le specie di animali più comuni che si possono trovare in uno zoo. Ad esempio, la *Casa delle Scimmie*.
- Esposizioni, in cui vivono alcune specie di animali particolari. Ad esempio, il *Rettilario*.
- Altre zone di servizio (in cui non vivono animali): ad esempio, la biglietteria, un bar, una zona picnic, alcuni bagni.

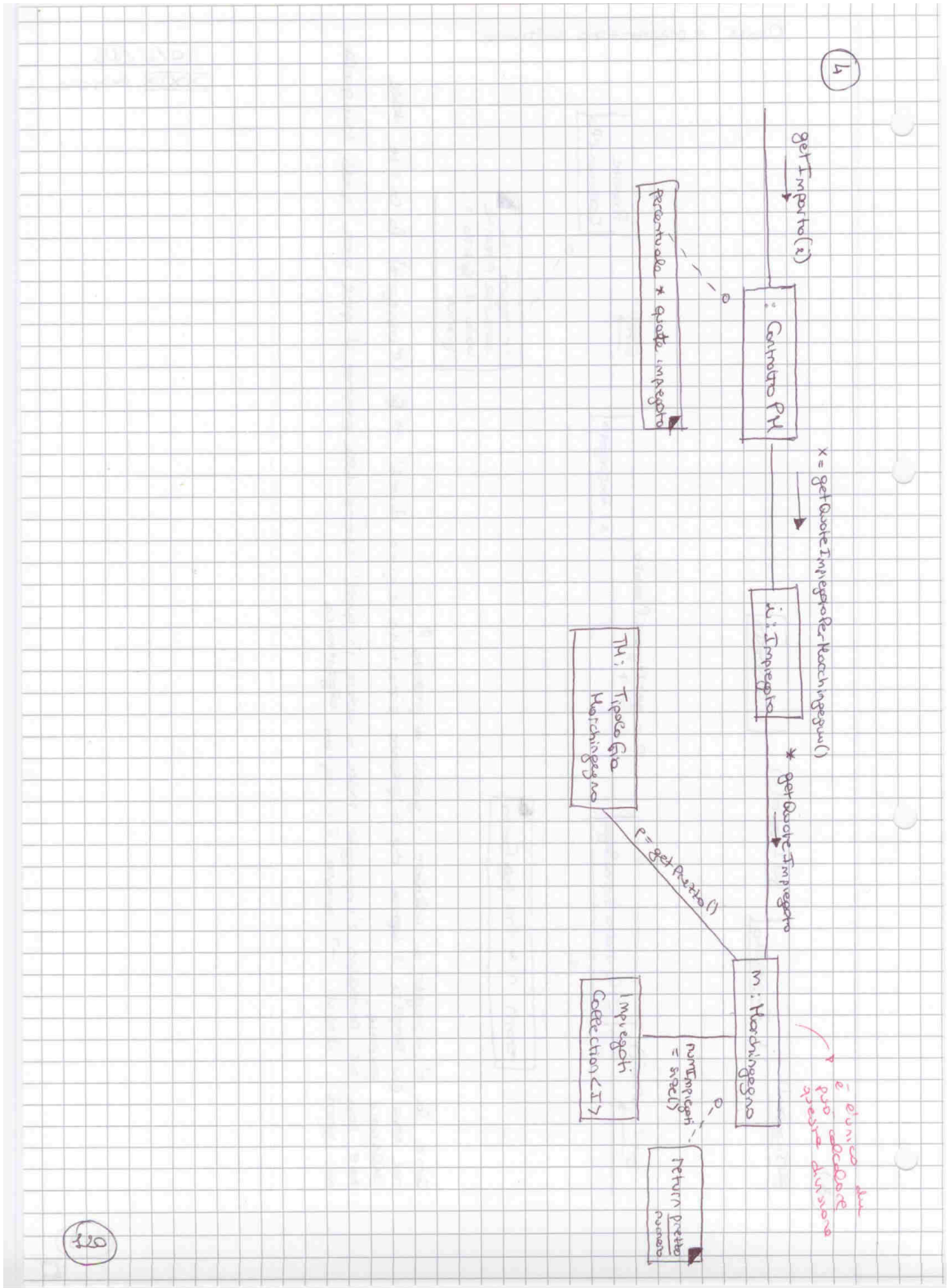
Le diverse aree dello zoo sono collegate tra di loro tramite dei tornelli, ovvero dei passaggi in una sola direzione, in cui può passare una sola persona per volta, utilizzando il proprio biglietto (si pensi ad esempio ai tornelli della metropolitana). Qui sotto è mostrata una porzione della mappa dello Zoo.



Il monitoraggio delle presenze dei visitatori nelle aree dello Zoo viene effettuato come segue:

- All'ingresso dello Zoo, a ciascun visitatore viene consegnato un biglietto (caso d'uso UC1).
- Nello Zoo, ogni volta che un visitatore vuole entrare in un'area, lo deve fare attraverso un tornello (vedi sopra: un tornello è un passaggio in una sola direzione tra due aree), utilizzando il proprio biglietto; ogni volta che un visitatore attraversa un tornello, il sensore del tornello invia al sistema di monitoraggio i dati sul transito rilevato (caso d'uso UC2).
- Quando un visitatore esce dallo Zoo, paga la visita in funzione del tipo di biglietto che ha scelto, delle aree dello zoo che ha visitato e del tempo che ha trascorso nelle diverse aree dello Zoo (caso d'uso UC3).

Osservazione: Nel progetto, è di interesse rappresentare la topologia dello Zoo, i nomi delle aree, il modo in cui sono utilizzate le diverse aree, nonché i transiti dei visitatori attraverso i tornelli e le presenze dei visitatori nelle aree dello Zoo. Invece, NON è di interesse per questa iterazione rappresentare quali specie di animali vivono nelle diverse aree, e nemmeno rappresentare i singoli animali.


4
getImporto(x)
PercentualeQuota * quota impiegato
ContrattoPM
x = getQuoteImpiegatoPerMacchinaGiusto()
xi : Impiegato
TM : TipologiaMacchinaGiusto
getQuotaImpiegato
r = getPartita()
M : Macchinagiusto
ImpiegatiCollection<IT>
numImpiegati = size()
return prezzo numero
è l'unico che può calcolare questa divisione
120

Analisi e progettazione del software
Anno Accademico 2012-2013
Zoo Acme – Requisiti

Ad esempio, prendiamo in considerazione un visitatore dello Zoo, Dario, e i suoi spostamenti all'interno dello Zoo:

- Dario alle 11:10 entra nello Zoo, con un suo biglietto, nella Casa delle Scimmie, attraverso il tornello 1.
- Dario alle 11:30 entra nella Casa delle Giraffe, attraverso il tornello 5.
- Dario alle 12:10 entra nel Rettilario, attraverso il tornello 21.
- Dario alle 12:50 entra nella Zona Picnic, attraverso il tornello 41.
- Dario alle 13:30 entra nella Casa degli Elefanti, attraverso il tornello 40.
- Dario alle 13:45 entra nella Casa delle Giraffe, attraverso il tornello 23.
- Dario alle 13:47 entra nella Zona Picnic, attraverso il tornello 46.
- Dario alle 13:48 esce dallo Zoo, attraverso il tornello 70.

Quando Dario arriva alla cassa, viene presentato il seguente resoconto dettagliato delle sue attività/presenze:

- Casa delle Scimmie, dalle 11:10 alle 11:30 (20 minuti)
- Casa delle Giraffe, dalle 11:30 alle 12:10 (40 minuti)
- Rettilario, dalle 12:10 alle 12:50 (40 minuti)
- Casa degli Elefanti, dalle 13:30 alle 13:45 (15 minuti)
- Casa delle Giraffe, dalle 13:45 alle 13:47 (2 minuti)

Il prezzo di una visita allo Zoo dipende dal tipo di biglietto scelto dal Visitatore, sulla base delle seguenti regole:

- **Biglietto bianco (a prezzo fisso)**: Il prezzo della visita allo Zoo è fissato a priori (11 euro).
- **Biglietto verde (a tempo)**: Il prezzo della visita è calcolato moltiplicando il numero di ore in cui il Visitatore è stato complessivamente presente nello Zoo (arrotondato per eccesso) per un fattore moltiplicativo fisso (3 euro l'ora).
- **Biglietto blu (a tempo di visita)**: Il prezzo della visita è calcolato moltiplicando il numero di ore in cui il Visitatore è stato presente in case di animali ed esposizioni dello Zoo (ignorando dunque il tempo trascorso nelle altre zone di servizio dello Zoo) per un fattore moltiplicativo fisso (4 euro l'ora).

Altri tipi di biglietto e regole di prezzo saranno prese in considerazione in iterazioni successive.

Ad esempio:

- Se Dario avesse scelto il biglietto bianco (a prezzo fisso), allora dovrebbe pagare 11 euro.
- Se Dario avesse scelto il biglietto verde (a tempo), allora dovrebbe pagare 9 euro (poiché è stato complessivamente presente nello Zoo per 2 ore e 38 minuti, arrotondate a 3 ore).
- Se Dario avesse scelto il biglietto blu (a tempo di visita), allora dovrebbe pagare 8 euro (poiché è stato presente in case di animali ed esposizioni dello Zoo per 1 ora e 57 minuti, arrotondate a 2 ore).

Alcune importanti osservazioni riguardanti l'analisi e la progettazione per questo sistema.

- Il monitoraggio delle presenze dei visitatori può essere gestito mediante la rappresentazione e memorizzazione dei *transiti* dei visitatori attraverso i tornelli dello Zoo, oppure mediante la rappresentazione e memorizzazione delle *presenze* dei visitatori nelle diverse aree. Il transito attraverso un tornello è istantaneo, mentre una presenza rappresenta un periodo di tempo trascorso in una certa area.
- Intuitivamente, le presenze possono essere derivate dai transiti. Ogni volta che un visitatore X transita da un'area A ad un'area B, allora la presenza in corso del visitatore X nell'area A si considera terminata, e inizia una nuova presenza del visitatore X nell'area B.
- **Nell'ANALISI, e in particolare nella modellazione di dominio, si chiede di rappresentare esplicitamente sia i TRANSITI che le PRESENZE.**
- **Nella PROGETTAZIONE, viceversa, si chiede di utilizzare una rappresentazione basata solo sulle PRESENZE (e non sui TRANSITI).**

Analisi e progettazione del software
Anno Accademico 2012-2013
Zoo Acme – Requisiti

L'uso del sistema in discussione è descritto principalmente dai seguenti casi d'uso UC1-UC3, di cui vengono riportati solo gli scenari principali:

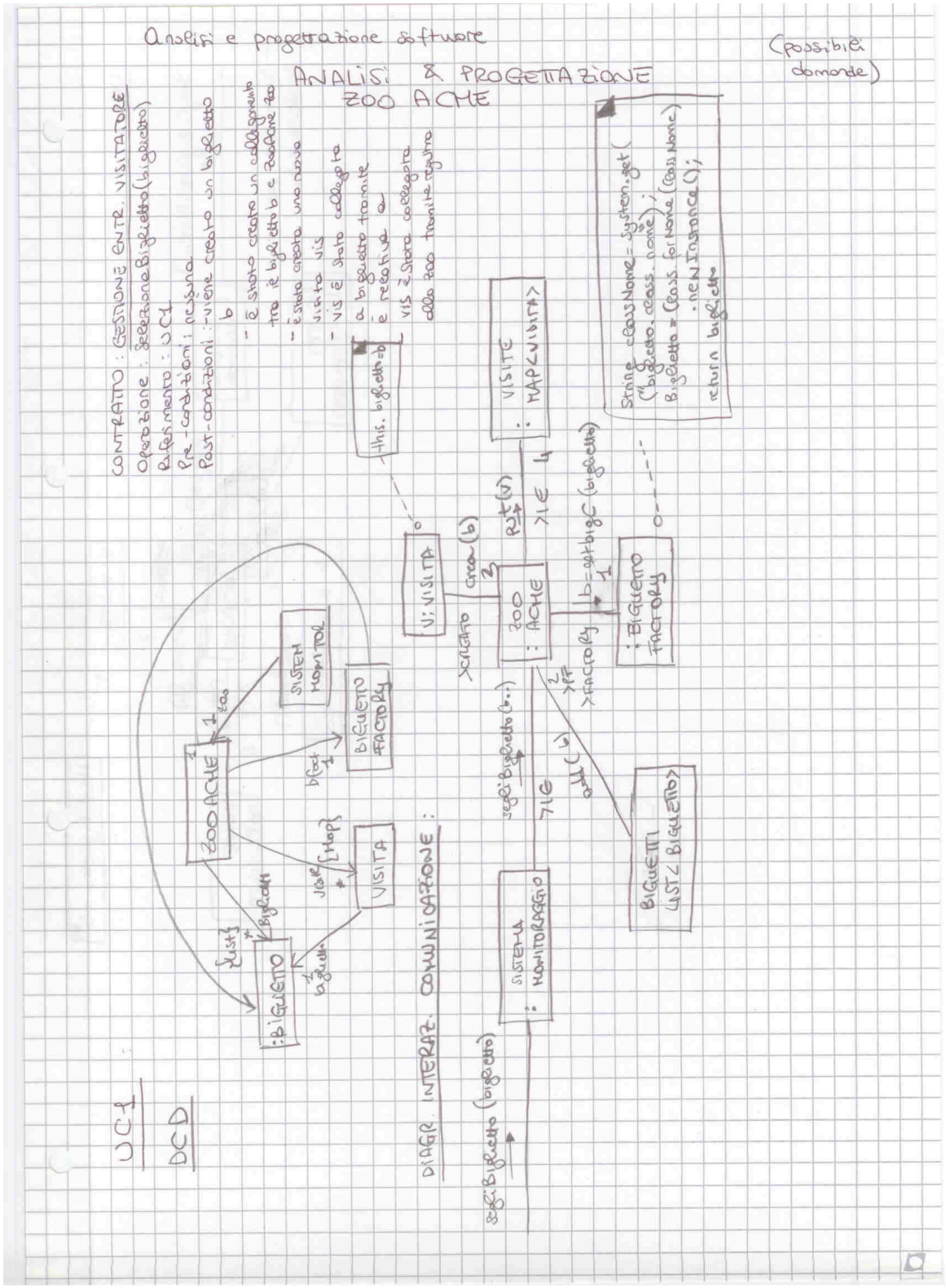
Caso d'uso UC1: Gestione entrata visitatore – Attore primario: un Cassiere.
Scenario principale di successo:
1. Un Visitatore si reca alla cassa dello Zoo perché vuole visitare lo Zoo.
2. Il Visitatore dice al Cassiere il tipo di biglietto che sceglie. (*I tipi di biglietti offerti dallo Zoo sono descritti nella pagina successiva.*) Il Cassiere inserisce il tipo di biglietto scelto dal Visitatore.
3. Il Sistema registra l'inizio di una nuova visita nello Zoo, il tipo di biglietto scelto dal visitatore, la data e l'ora d'entrata, e predispone un biglietto individuale per il Visitatore (sulla cui banda magnetica viene registrato un codice identificativo univoco per la visita).
4. Il Cassiere consegna il biglietto al Visitatore, che lo prende e inizia la sua visita.

Osservazione: *Nell'iterazione corrente, sono di interesse solo visitatori individuali.*

Caso d'uso UC2: Gestione transito visitatore – Attore primario: un Tornello dello Zoo.
Scenario principale di successo:
1. Un Visitatore vuole passare in un'altra area dello Zoo, attraversando un tornello.
2. Il Visitatore inserisce il proprio biglietto nel tornello e lo attraversa. Il Tornello invia al Sistema un messaggio composto da: (i) codice identificativo del tornello e (ii) codice identificativo del biglietto del Visitatore.
3. Il Sistema registra il transito del Visitatore in modo opportuno, memorizzando anche data e ora del transito.

Osservazione: *Il caso d'uso UC2 viene eseguito in modo continuo da tutti i tornelli presenti nello Zoo. Ciascun tornello conosce il proprio codice identificativo ed è in grado di leggere dalla banda magnetica del biglietto il codice identificativo associato al visitatore in transito.*

Caso d'uso UC3: Gestione uscita visitatore – Attore primario: un Cassiere.
Scenario principale di successo:
1. Un Visitatore si reca alla cassa dello Zoo, per pagare la visita.
2. Il Visitatore consegna il proprio biglietto al Cassiere.
3. Il Cassiere inserisce il codice identificativo del biglietto del Visitatore.
4. Il Sistema mostra l'elenco dettagliato delle attività del Visitatore, mostrando per ciascuna presenza del Visitatore in una delle aree dello Zoo in cui vivono animali: l'area visitata, l'orario d'ingresso nell'area, l'orario di uscita dall'area, la durata della presenza nell'area. (Il Sistema non mostra, invece, nessuna informazione sulle presenze del Visitatore nelle altre zone di servizio dello Zoo, come la zona picnic e il bar.) (*Un esempio è mostrato nella pagina successiva.*)
5. Il Sistema calcola il prezzo per la visita dello Zoo, in base a un insieme di regole di prezzo. (*Le regole di prezzo sono descritte nella pagina successiva.*)
6. Il Visitatore paga (in contanti). Il Sistema registra il pagamento e genera una ricevuta.


Analisi e progettazione software
ANALISI & PROGETTAZIONE
ZOO ACHE
(possibili domande)
CONTRATTO : GESTIONE ENTR. VISITATORE
Operazione : accessoBiglietto(biglietto)
Riferimento : UC1
Pre-condizioni : nessuna
Post-condizioni : - viene creato un biglietto b
- è stato creato un collegamento tra il biglietto b e zoo ache
- è stata creata una nuova visita vis
- vis è stata collegata a biglietto tramite e relativa a vis è stata collegata allo zoo tramite registra
UC1
DCD
DIAGR. INTERAZ. COMUNICAZIONE :
ZOO ACHE
SISTEMA MONITORAGGIO
BIGLIETTO FACTORY
VISITA
BIGLIETTI
:BIGLIETTO
V: VISITA
VISITE
: MAP<Visita>
: BIGLIETTO FACTORY
BIGLIETTI LIST< BIGLIETTO>
crea (b)
associa
entra
esci
accessoBiglietto (b...)
getBiglietto (biglietto)
String className = System.get(
("biglietto.class.name");
Biglietto = (Class.forName(className)
.newInstance();
return biglietto

Analisi e progettazione del software
Anno Accademico 2012-2013
Progetto associato alla prova intermedia del 5 giugno 2013

Regole: **SCRIVERE IN MODO LEGGIBILE E NON AMBIGUO**. Scrivere il proprio nome su ciascun foglio utilizzato, in alto a destra. Accanto allo svolgimento di ciascun esercizio o domanda va indicato chiaramente l'esercizio o la domanda a cui lo svolgimento è relativo (ad esempio, Esercizio X1, diagramma delle classi di progetto, oppure Esercizio X2, domanda X2.1). **Diagrammi accanto a cui non è indicato l'esercizio a cui il diagramma è relativo non saranno corretti**.

Ipotesi di lavoro, valide per tutti gli esercizi di progettazione.

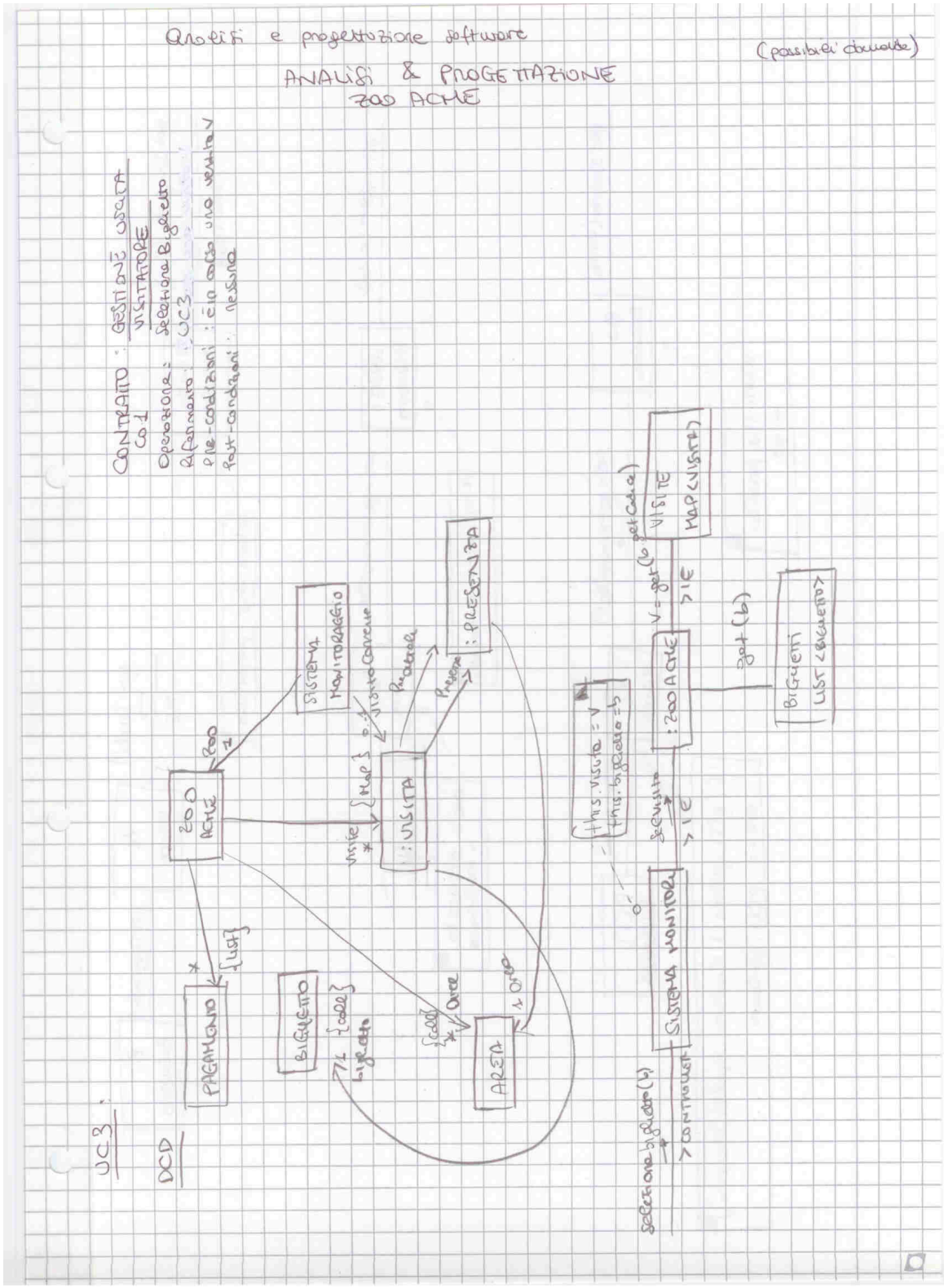
- In tutti gli esercizi che seguono, si faccia l'ipotesi che il sistema in discussione gestisca i propri dati solo in memoria principale. Si supponga anche che durante il caso d'uso di avviamento vengano creati e caricati in memoria tutti gli oggetti le cui informazioni siano già effettivamente disponibili al momento dell'avviamento.
- Per ciascuna operazione di sistema va creato un diagramma di interazione che descrive l'interazione relativa alla trasformazione (cambiamento di stato) provocata dall'operazione di sistema. Per quanto riguarda invece le relative risposte (interrogazioni) eventualmente restituite dal sistema, se non è richiesto esplicitamente allora non bisogna mostrare nei diagrammi di interazione né il calcolo dei dati da restituire né la loro visualizzazione. Tuttavia, per le risposte del sistema, è comunque necessario verificare che i dati da restituire possano essere (facilmente) calcolati sulla base delle navigabilità tra gli oggetti che sono state progettate (vedi anche il punto successivo).
- **Le soluzioni individuate dovranno essere compatibili (in particolare in termini di visibilità, ovvero di navigabilità delle associazioni) con le realizzazioni di tutti i casi d'uso mostrati (UC1-UC3).**
- **Nei diagrammi di interazione, mostrare IN MODO ESPLICITO: tutti i MESSAGGI scambiati tra oggetti, tutte le CREAZIONI di oggetti e tutte le FORMAZIONI e ROTTURE di collegamenti.**
- Nei diagrammi di interazione, motivare le scelte di progetto fatte indicando i pattern GRASP e GoF applicati.
- Nei diagrammi delle classi di progetto, mostrare: (1) per ciascuna classe: il nome della classe, i nomi dei suoi attributi, i nomi delle sue operazioni; e (2) per ciascuna associazione e ciascuna sua estremità navigabile: la freccia di navigabilità, il nome dell'estremità, la molteplicità e, in caso di associazione navigabile a molti, il tipo di collezione scelta.

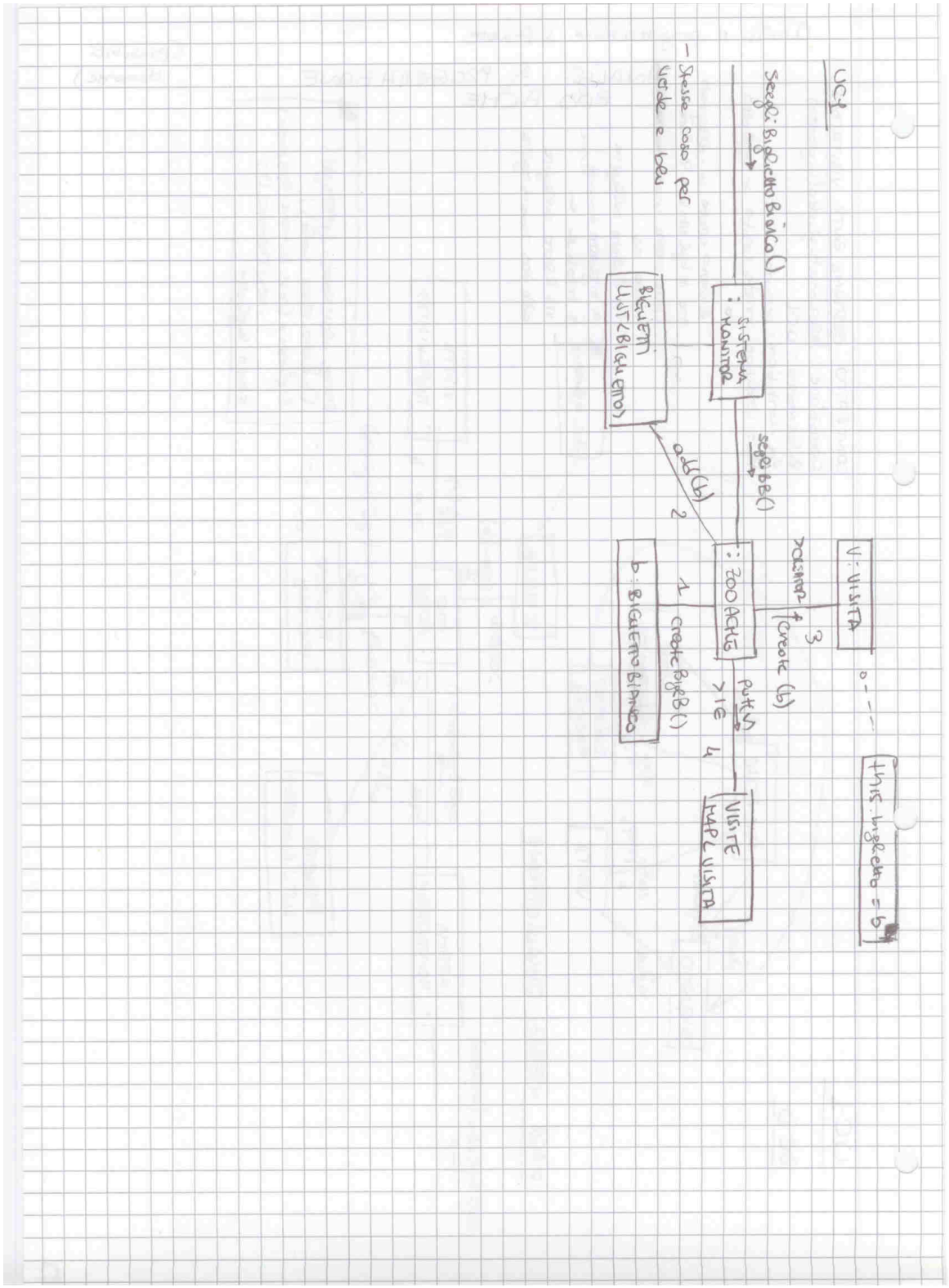
Esercizio P1 (ANALISI)
Fare l'analisi orientata agli oggetti per il sistema in discussione, relativamente a tutti i casi d'uso mostrati, come segue:

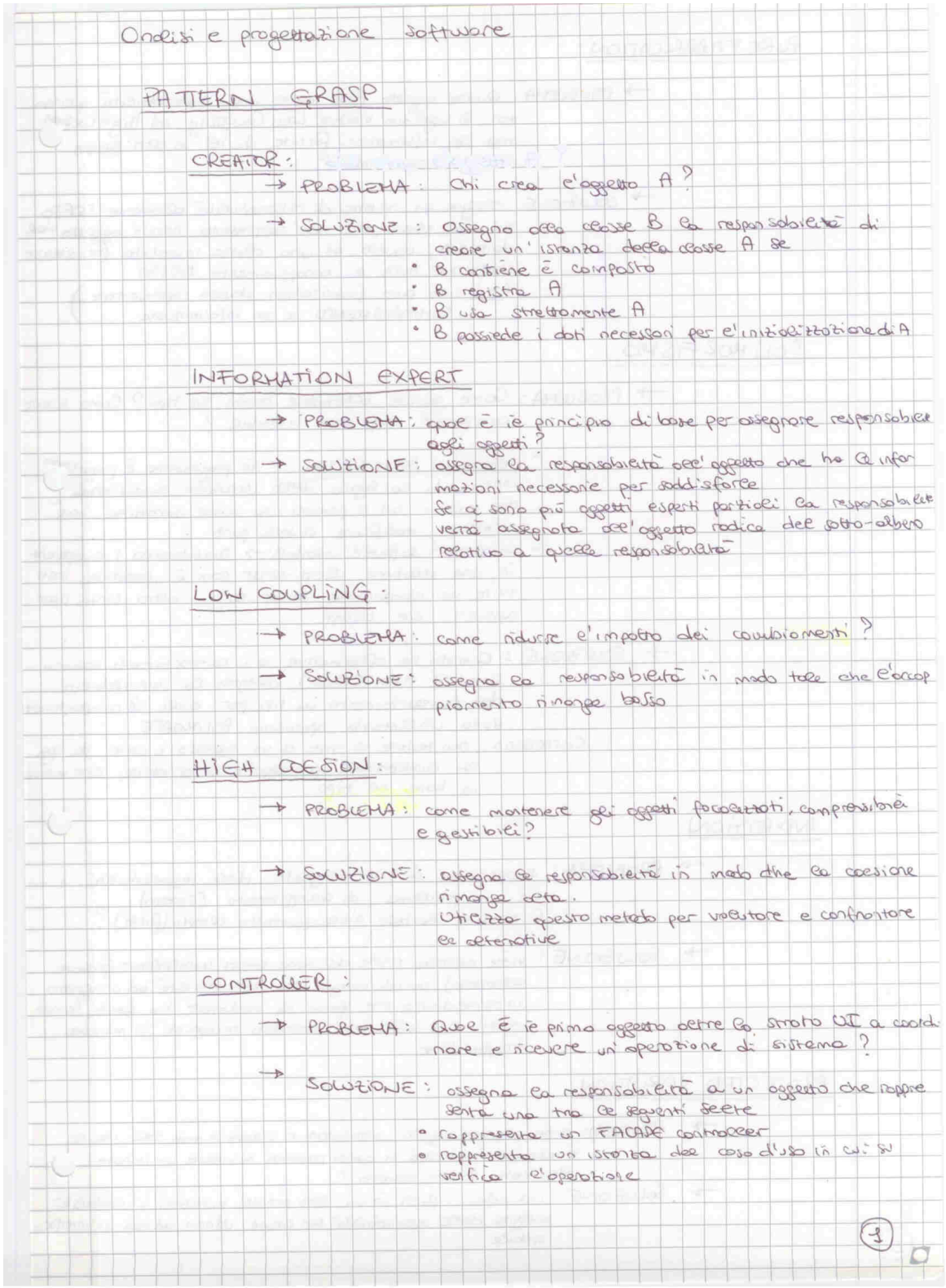
- Mostrare il modello di dominio.
- Mostrare un diagramma di oggetti di dominio che rappresenta:
  - La Zona Picnic, la Casa degli Elefanti, la Casa delle Giraffe, i tornelli 40, 23 e 46 (si veda la figura).
  - Una visitatrice (Benedetta), che ha scelto il biglietto blu (a tempo di visita).
  - Il transito di Benedetta dalla Zona Picnic alla Casa degli Elefanti alle 16:00.
  - Il transito di Benedetta dalla Casa degli Elefanti alla Casa delle Giraffe alle 16:30.
  - Il transito di Benedetta dalla Casa delle Giraffe alla Zona Picnic alle 17:00.
  - Tutte le presenze implicate dai transiti precedenti.

Esercizio P2 (PROGETTAZIONE)
Fare la progettazione orientata agli oggetti per il sistema in discussione, relativamente al caso d'uso UC2 (*Gestione transito visitatore*), come segue:

- Mostrare il diagramma di interazione relativo all'unica operazione di sistema *transito(idTornello, idBiglietto)* che può essere identificata nel caso d'uso UC2.
- Mostrare il diagramma delle classi di progetto corrispondente.


Analisi e progettazione software
ANALISI & PROGETTAZIONE
ZOO ACME
(possibili domande)
CONTRATTO: GESTIONE VISITA
Cod
VISITATORE
Operazione: selezionaBiglietto
Riferimento: UC3
Pre-condizioni: è in atto una sessione
Post-condizioni: nessuna
UC3:
DCD
PAGAMENTO
BIGLIETTO
ZOO ACME
SISTEMA MONITORAGGIO
: VISITA
: PRESENTAZIONE
: ZOO ACME
SISTEMA MONITORAGGIO
BIGLIETTI
LIST <BIGLIETTO>
MAP<VISITE>
VISITE


UC1
- Stesso caso per verde e blu
: SISTEMA MONITOR
: ZOOARCH
: BIGLIETTI LIST<BIGLIETTO>
b: BIGLIETTO BIANCO
V: VISITA
this.biglietto = b
: VISITE MAP<V,LISTA>
1
2
3
4



Analisi e progettazione software

PATTERN GRASP

CREATOR:

→ PROBLEMA: Chi crea l'oggetto A?

→ SOLUZIONE: Assegno alla classe B la responsabilità di creare un'istanza della classe A se
- B contiene è composto
- B registra A
- B usa strettamente A
- B possiede i dati necessari per l'inizializzazione di A

INFORMATION EXPERT

→ PROBLEMA: Qual è il principio di base per assegnare responsabilità agli oggetti?

→ SOLUZIONE: Assegna la responsabilità dell'oggetto che ha le informazioni necessarie per soddisfarla
Se ci sono più oggetti esperti parziali la responsabilità verrà assegnata dell'oggetto radice del sotto-albero relativo a quella responsabilità

LOW COUPLING:

→ PROBLEMA: Come ridurre l'impatto dei cambiamenti?

→ SOLUZIONE: Assegna la responsabilità in modo tale che l'accoppiamento rimanga basso

HIGH COESION:

→ PROBLEMA: Come mantenere gli oggetti focalizzati, comprensibili e gestibili?

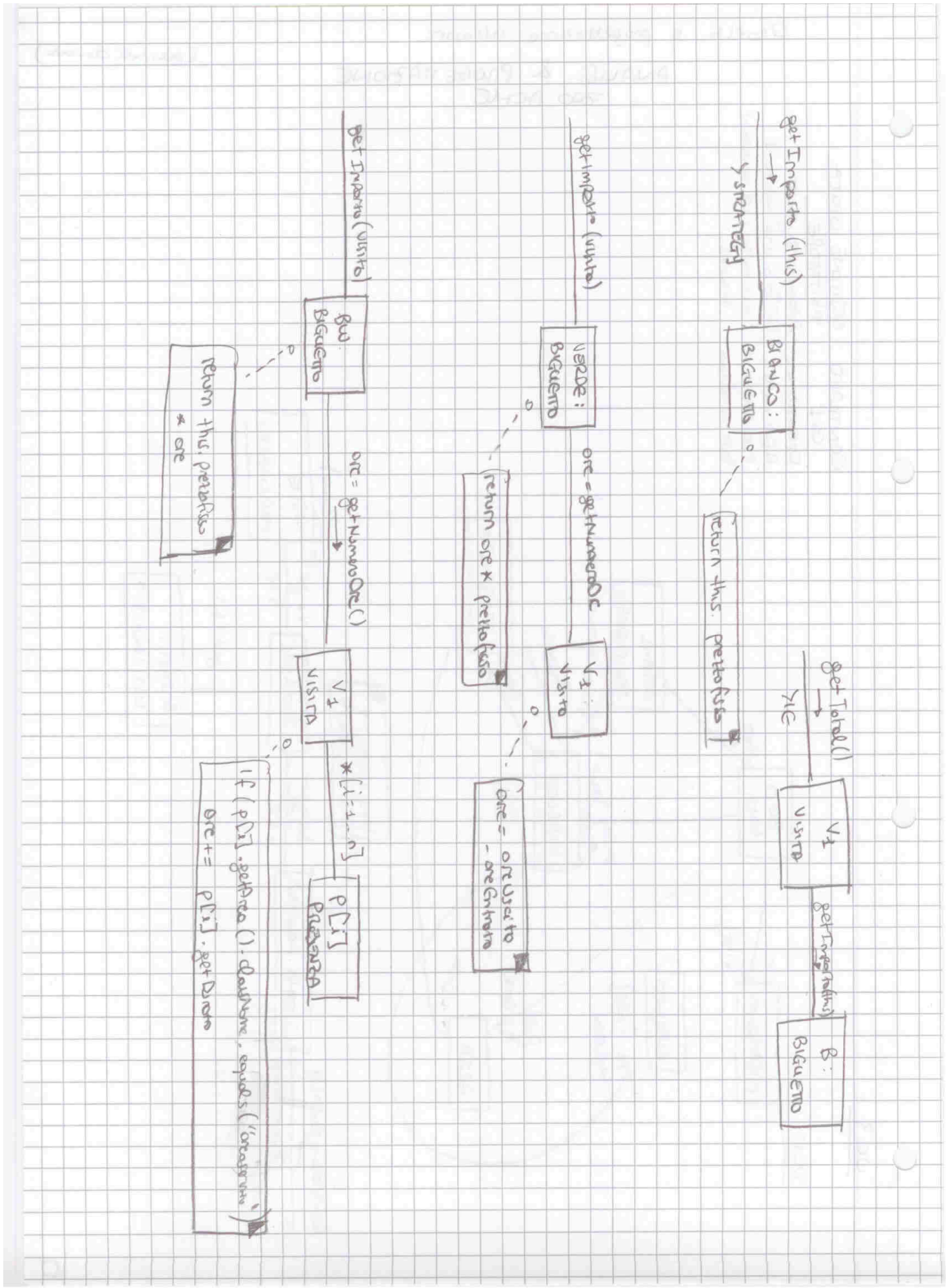
→ SOLUZIONE: Assegna la responsabilità in modo che la coesione rimanga alta.
Utilizza questo metodo per valutare e confrontare le alternative

CONTROLLER:

→ PROBLEMA: Qual è il primo oggetto oltre lo strato UI a coordinare e ricevere un'operazione di sistema?

→ SOLUZIONE: Assegna la responsabilità a un oggetto che rappresenta una tra le seguenti scelte
- rappresenta un FACADE controller
- rappresenta un istanza del caso d'uso in cui si verifica l'operazione

Sorgente : http://holsi.hasanaj.com/ Corso di Ingegneria Informatica Roma Tre
Autore : Holsi Hasanaj Corso: Analisi e Progettazione del Software
Professore: Luca Cabibbo Anno di produzione: 2012 - 2013



Analisi e progettazione del software

PATTERN GOF:

ADAPTER:

→ PROBLEMA: come gestire interfacce incompatibili o fornire un' interfaccia stabile a componenti simili ma con interfacce diverse
- interfacce INCOMPATIBILI: consideriamo coppia oggetti SW in relazione client-server. L'oggetto server offre dei servizi di interesse per l'oggetto CLIENT, tuttavia l'oggetto client vuole fruire di questi servizi in una modalità diversa da quella prevista dall'oggetto server
- componenti simili con interfacce diverse: un oggetto client vuole fruire dei servizi offerti da uno tra più oggetti server che offrono servizi simili

→ SOLUZIONE: usando un intermediario tra il client e il servizio che ha lo scopo di convertire, l'interfaccia del servizio in un altra interfaccia gradita al CLIENT sulla base dell'oggetto inferito
(n.b: adapter consente a classi diverse di operare insieme quando ciò non sarebbe altrimenti possibile a causa di interfacce incompatibili)

FACTORY:

→ PROBLEMA: Chi deve essere responsabile della creazione di oggetti quando ci sono delle considerazioni speciali, come una logica di creazione complessa, quando si desidera separare la responsabilità di creazione per una coesione migliore e così via?

→ SOLUZIONE: definisci un oggetto pure Fabrication chiamato Factory "fabbrica" che gestisce la creazione
- fornire un interfaccia per la creazione di un oggetto senza specificare quale sia la sua classe concreta

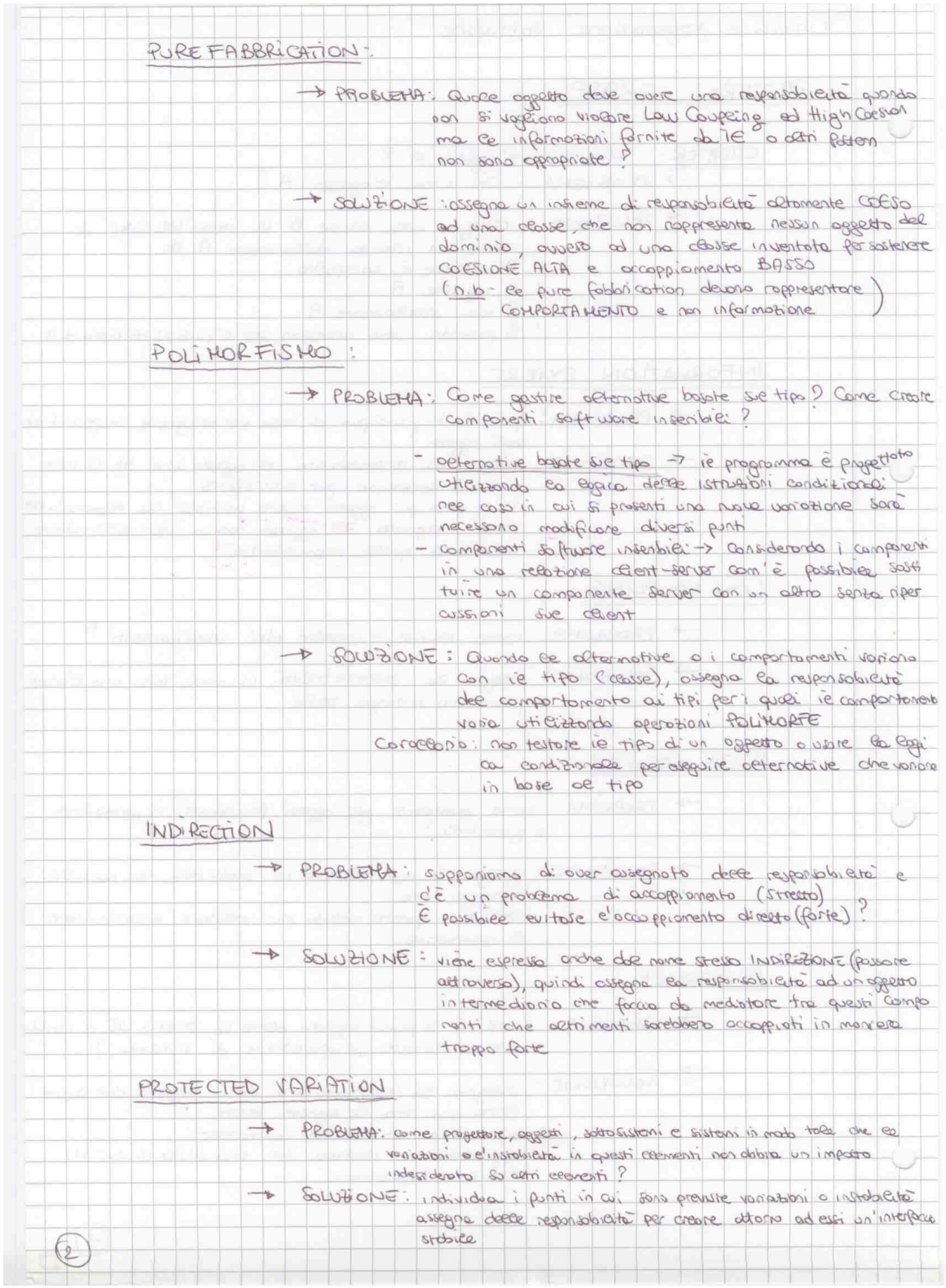
SINGLETON:

→ PROBLEMA: è consentita esattamente una sola istanza di una classe ovvero un singleton. Gli oggetti hanno bisogno di un p.to di accesso globale e singolo
(Chi crea la factory e come ci si accede?)

→ SOLUZIONE: definisci un metodo statico della classe che restituisce il singleton
- lo scopo è quello di fornire un p.to di accesso globale alla factory per creare adapter per l'accesso ai servizi

STRATEGY:

→ PROBLEMA: - come progettare per gestire un insieme di algoritmi (al meno 2) o politiche variabili ma correlate?
- come progettare per consentire di modificare quest' ultimi algoritmi o politiche?



PURE FABBRICATION:

→ PROBLEMA: Quale oggetto deve avere una responsabilità quando non si vogliono violare Low Coupling ed High Cohesion ma le informazioni fornite da IE o altri pattern non sono appropriate?

→ SOLUZIONE: assegna un insieme di responsabilità altamente COESO ad una classe, che non rappresenta nessun oggetto del dominio, ovvero ad una classe inventata per sostenere COESIONE ALTA e accoppiamento BASSO
(n.b. le pure fabbrication devono rappresentare COMPORTAMENTO e non informazione)

POLIMORFISMO:

→ PROBLEMA: Come gestire alternative basate sul tipo? Come creare componenti software inseribili?

- alternative basate sul tipo → il programma è progettato utilizzando la logica delle istruzioni condizionali nel caso in cui si presenti una nuova variazione sarà necessario modificare diversi punti
- componenti software inseribili → considerando i componenti in una relazione client-server com'è possibile sostituire un componente server con un altro senza ripercussioni sul client

→ SOLUZIONE: Quando le alternative o i comportamenti variano con il tipo (classe), assegna la responsabilità del comportamento ai tipi per i quali il comportamento varia utilizzando operazioni POLIMORFE
Conseguenza: non testare il tipo di un oggetto o usare la logica condizionale per eseguire alternative che variano in base al tipo

INDIRECTION

→ PROBLEMA: supponiamo di aver assegnato delle responsabilità e c'è un problema di accoppiamento (stretto)
È possibile evitare l'accoppiamento diretto (forte)?

→ SOLUZIONE: viene espresso anche dal nome stesso INDIREZIONE (passare attraverso), quindi assegna la responsabilità ad un oggetto intermediario che faccia da mediatore tra questi componenti che altrimenti sarebbero accoppiati in maniera troppo forte

PROTECTED VARIATION

→ PROBLEMA: come progettare, oggetti, sottosistemi e sistemi in modo tale che le variazioni o l'instabilità in questi elementi non abbia un impatto indesiderato su altri elementi?

→ SOLUZIONE: individua i punti in cui sono previste variazioni o instabilità assegna delle responsabilità per creare attorno ad essi un'interfaccia stabile

→ SOLUZIONE: definisci ciascun algoritmo / politica / strategia in una classe diversa e tutte queste classi implementano un' interfaccia comune
- Scopo → definire una famiglia di algoritmi, incapsularli e renderli intercambiabili

Dietro agli oggetti strategia nella soluzione è presente un altro oggetto che prende il nome di: OGGETTO CONTESTO ed è proprio l'oggetto sul quale va applicato l'algoritmo → parametro della strategia

OBSERVER:

→ PROBLEMA: - diversi tipi di oggetti subscriver (abbonato) sono interessati ai cambiamenti di stato o agli eventi di un oggetto publisher
- ciascun subscriver vuole reagire in modo proprio quando un publisher genera un evento
- il publisher vuole mantenere un accoppiamento basso verso i suoi subscriver

→ SOLUZIONE: definisci un interfaccia "subscriver" o "listner"
i subscriver implementano quest' interfaccia ascoltatore
Il publisher può registrare dinamicamente i subscriver che sono interessati ai suoi eventi e avvisarli quando si verifica un evento (oggettificare gli eventi → viene fatto attraverso interfaccia)
- Scopo → definire una dipendenza uno a molti fra oggetti in modo tale che se un oggetto cambia il suo stato, tutti gli oggetti dipendenti da questo siano notificati e aggiornati automaticamente

PROXY:

→ PROBLEMA: c'è un client ed un server e l'accesso del servizio da parte del client al server è possibile ma si potrebbe migliorare ① sicurezza ② efficienza ③ prestazioni

PROXY VS ADAPTER: Proxy è diverso da adapter perché qui l'interfaccia del proxy è proprio quella del servente mentre in adapter non è così a causa di un incompatibilità di interfaccia

→ SOLUZIONE: introdurre un intermediario in modo tale che quando il client vuole accedere al server, in verità accede al proxy e poi è il proxy che accede al server al posto del client
Un aspetto importante → quando il proxy accede al server, prima dell' accesso al server e dopo l'accesso al server il servizio può eseguire delle pre-elaborazioni e dopo post-elaborazioni

FACADE:

FACADE VS ADAPTER: sono diversi perché → adapter ce ne possono essere molte, mentre con facade c'è implementazione è una sola. Inoltre adapter è legato ad accesso di cose pre-esistenti, mentre facade è legato a cose di cui non si sa come sono fatte o come devono essere fatte

→ PROBLEMA: - è richiesta un' interfaccia comune e unificata per un insieme disparato di implementazioni o interfacce.
- Può verificarsi un accoppiamento indesiderato a molti oggetti nel sottosistema oppure l'implementazione del sottosistema può cambiare

→ SOLUZIONE: definisci un p.to di contatto singolo con il sotto-sistema, ovvero una facciata
- quest'oggetto facciata presenta un interfaccia singola e unificata ed è responsabile della collaborazione con i componenti del sotto-sistema

TEMPLATE:

→ PROBLEMA: come progettare per gestire un insieme di algoritmi correlati che hanno tutti la stessa struttura ma variano nelle implementazioni di alcuni passi?

→ SOLUZIONE: definisci l'algoritmo e la struttura come metodo in una superclasse
- in questa classe definisci un metodo eventualmente astratto per ciascuno dei passi dell'algoritmo e infine mentre l'algoritmo sulla base delle invocazioni di questi metodi
- per ciascuna versione dell'algoritmo definisci una sottoclasse concreta della superclasse che definisce l'implementazione specifica dei passi