Verso la progettazione ad oggetti

Requisiti=”Cosa fare?”

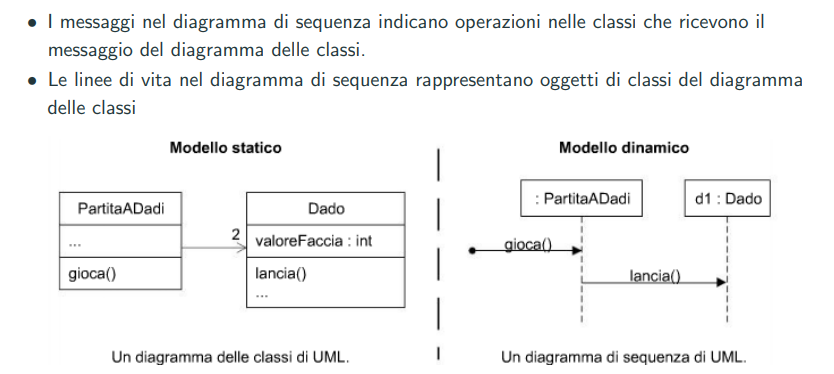
Progettazione=”Come lo si fa bene?”

Modellazione agile: modellare per comprendere e comunicare più che per documentare.

Modelli dinamici: la modellazione a oggetti dinamica più comune eè qquella con diagrammi di sequenza di UML. Rappresenano il comportamento, la collaborazione tra gli oggetti per realizzare un caso d’uso.

Modelli statici: la modellazione a oggetti statica più comune è quella con i diagrammi delle classi UML. Definisce i package, nomi e attributi delle classe, firme delle operazioni.

I due modelli sono relazionati tra loro, si consiglia di crearli quindi in parallelo



Diagrammi di interazione

Interazione: specifica di come oggetti si scambiano messagg nel tempo al fine di eseguire un compito. Un compito è rappresentato da un messaggio invato da un oggetto, designato come “responsabile” del com’puto. Ogni partecipante all’interno dell’interazione svolge un suo ruolo.

Pro: chiarezza nella sequenza temporale dei messaggi

Contro: se l’interazione coinvolge tanti oggetti, occupa tanto spazio orizzontalmente

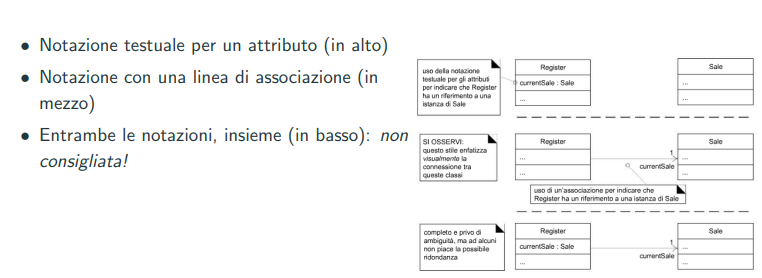
Design Sequence Diagram (DSD): diagramma di sequenza utilizzato dal punto di vista software/progettistico. iI UP l’’insieme di tutti i DSD fanno parte del modello di progetto

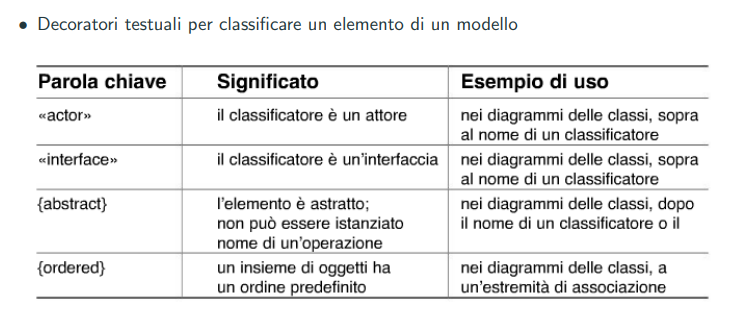
Singleton: pattern nel quale una classe viene istanziala una e una sola volta

Come leggerli e crearli scritto nelle slide

Diagrammi delle classi

UML che comprende i diaagrammi delle classi per illustrare classi, interfacce e associazioni, utilizzato per visualizzare concettualmente un modello di dominio





Definizione responsabilità OOD (Object Oriented Design)

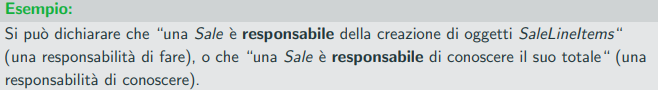
Definizione dei metodi delle classi e i messaggi che queste di scambiano tra loro per soddisfare i requisiti ⇒ vengono applicati principi di progettazione OO (E.G. GRASP, GoF)

Responsibility Driven Development (RDD): approccio alla modellazione e progettazione OO basata sull’assegnamento delle responsabilità e dalle collaborazioni tra oggetti.

In UML la responsabilità è un contratto o un obbligo di un classificatore

Le responsabilità sono di due tipi:

* Di fare
  + Creare un oggetto, eseguire un calcolo
  + Chiedere ad altri oggetti di eseguire azioni
  + Controllare e coordinare le attività di altri oggetti
* Di conoscere
  + Conoscere i propri dati privati
  + Conoscere gli oggetti correlati
  + Conoscere i dati che può derivare o calcolare



A seconda della granularità delle responsabilità, questa può coinvolgere centinaia di classi o solo un metodo.

La progettazione RDD viene fatta iterativamente seguenti i passi:

* identificazione delle responsabilità
* Identificazione dell’oggetto a cui assegnarla, eventualmente un nuovo oggetto
* Identificazione delle azioni che l’oggetto deve svolgere per soddisfare la responsabilità, identificando anche possibili altri oggetti con cui deve collaborare

General Responsibility Assignment Software Pattern (GRASP)

Principi e pattern utili per aiutare nell’apprendimento degli aspetti progettistici ad oggetti, fornisce ragionamenti, meodoto struttura e spiegabilità.

L’obiettivo è quello di avere un software facile da capire, mantener, estendere e riusare⇒progettazione modulare

L’assegnazione delle responsabilità si possono prendere durante la modellazione o mentre si codifica

Pattern: principi ed idiomi che, codificati in un formato strutturato, descrive problema ricorrente e la sua soluzione ⇒ possono essere applicati a contesti nuovi con compromessi o variazioni.

In GRASP i pattern sono 9:

1. Creator
   1. Problema: Chi crea l’oggetto A? Di chi è la responabilità di istanziare un nuovo oggetto?
   2. Soluzione: la classe B è responsabile di creare l’istanza di A se:
      1. B “contiene” oggtti della classe A
      2. B registra A^2
      3. B utilizza strettamente A
      4. B possiede dati necessari all’inizializzazione di A
2. Informatione expert:
   1. Problema: secondo quale principio assegniamo le responabilità?
   2. Soluzione: assegna la responsabilità alla classe che possiede le informazioni necessarie per soddisfarla (l’esperto dlele informazioni). Alcune informazioni necessarie per decidere a chi assegnare la responsabilità, possono essere distribuite tra più classi, in questo caso queste collaborano insieme per realizzare l’obiettivo
3. Low coupling:
   1. Problema: come ridurre l’impatto dei cambiamenti? Come mantenere basse dipendenze in modo da favore il riuso?
   2. Soluzione: assegna le responabilità in modo che l’accoppiamento rimanga il più basso possibile. Accoppiamenti alti (o forti) implicano forti dipendenze tra classi, quindi bassa indipendenza tra i vari moduli con i conseguenti inconvienienti nel mantenimento e nel riutilizzo, e bassa chiarezza locale, ovvero difficile comprensione delle classi in isolamento, senza comprendere le classi da cui dipendono
4. High cohesion
   1. Problema: mantenere gli oggetti focalizzati, omprensibili e gestibili, sostenendo comunque il pattern Low Coupling?
   2. Soluzione: assegnare le responsabilità in modo che la coesione funzionale sia alta. La coesione è la misura di quanto forte siano correlate le responsabilità di un elemento. Alta coesione significa che un elemento è altamente correlato ma non esegue una quantità di lavoro eccessiva. Una basse coesione, al contrario, implica che una classe gestisca diverse cose non correlate tra loro o svolge una quantità di lavoro eccessiva ⇒ difficile da capire, mantenere e riusare. Classi a coesione bassa suggeriscono che alcune responsabilità dovrebbero esser state delegate ad altri oggetti. Una classe ad alta coesione ha un numero di metodi relativamente basso, con funzionalità altamente focalizzate sulle proprie responsabilità. La coesione può essere valutata secondo diverse metriche:
      1. Coesione dati: una classe gestisce un tipo di dato
      2. Coesione funzionale:gli elementi di una classe svolgono una singola funzione
      3. Coesione temporale: gli elementi sono raggruppati perchè attivati circa nello stesso tempo (a volte poco efficente)
5. Controller
   1. Problema: identificare il primo oggetto (UI esclusa) che riceve e coordina (“controlla”) un’operazione di sistema?
   2. Soluzione:“controller” è semplicemente un pattern di delega, dove il controller delega il lavoro ad oggetti di un alto strato. Assegnamento delle responsabilità ad un oggetto secondo una delle seguenti idee:
      1. l’oggetto rappresenta il “sistema complessivo”, un “oggetto radice”, un ounto di accesso al software o un sottosistema principale
      2. rappresenza uno scenario di un caso d’uso all’interno del quale si verifica l’operazione di sistema (“controller di casod’uso”)