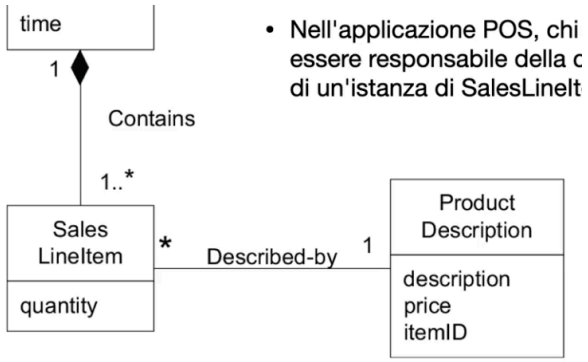
GRASP

Thursday, 25 May 2023 14:38

- Stiamo parlando della progettazone orientata agli oggetti
  Input:
  - Testo casi d'uso
  - Specifiche supplementari (requisiti non funzionali)
  - Glossario
  - Diagrammi di sequenza di sistema
  - Modello di dominio (si prende ispirazione)
  - Contratti delle operazioni (mostrano le iterazioni degli oggetti software per soddisfare vincoli)
- Attività per la creazione del modello di progetto:
  Abbiamo 3 possibilità:
  - Progettare mentre si codifica (sviluppo guidato dai test)
    - Si creano i test affinchè possiamo essere sicuri che, ciò che noi andiamo a creare non vadi ad influenzare parti passate
  - Iniziare modellazione UML
  - Schede CRC

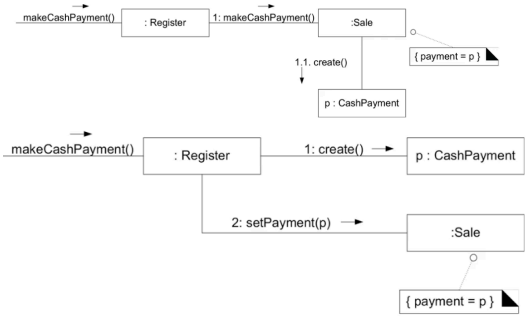
  Ed in tutte e 3 vengono applicati:
  - Pattern GRASP
    - Danno un nome ed una descrizione di principi base per la progettazione di oggetti ed assegnazione di responsabilità
    - I suoi pattern:
      - Creator
        - Problema: chi crea oggetto A?
        - Soluzione: Assegna a classe B la creazione di classe A se una delle seguenti soluzioni è vera:
          - B contiene o aggrega composizione oggetti A
          - B registra A
          - B utilizza strettamente A
          - B possiede dati per l'inizializzazione di A
        - Se queste vengono assegnate bene, abbiamo un accoppiamento basso, riusabilità e maggiore chiarezza
          Qui sales dovrebbe essere il creator di saleItem



- Information expert
  - Problema: Qual'è il principio per assegnare responsabilità agli oggetti?
  - Soluzione: chi ha le informazioni necessarie per soddisfarla
  - Spesso ci sono molti esperti parziali
    Si assegnano responsabilità alle persone che posseggono le informazioni necessarie per svolgere un compito
- Low coupling
  - Problema: come ridurre l'impatto dei cambiamenti?
  - Soluzione: Assegnare le risponsabilità in modo tale che l'accoppiamento rimanga basso
    Aka diminuire le dipendenze
    Mettiamo caso abbiamo: classe Cane, classe tavolo e classe caselle
    Noi siamo Cane
    Noi chiediamo a tavolo "dammi tutte le caselle"
    E poi, noi iteriamo per tutte le caselle alla ricerca della casella con nome 'inutile'.
    Qui noi abbiamo 2 dipendenze: tavolo->casella, cane->casella
    E' possibile diminuirlo dando la responsabilità di "dammi casella inutile" a tavolo, quindi chiamando una funzione di tavolo, e quindi togliendo la dipendenza cane->casella

    L'accoppiamento = la misura di quanto un elemento è collegato/conosce o si basa su altri elementi
    Il problema è che, se siamo collegati a 10 classi, alla modifica di 1 di queste 10 classi anchio sono da modificare. In più siccome si influenzano, è difficile da comprendere da isolamento e da riusare per il

da comprendere da isolamento e da riusare per il futuro.



Nel primo caso abbiamo un accoppiamento più basso rispetto la seconda immagine
Quindi se possibile avere il numero più possibile di classi indipendenti
Nota: se ci deve essere un accoppiamento, è meglio averlo con elemetni stabili = che cambiano poco nel tempo = ha un impatto basso

- High cohesion

Prima di tutto, cos'è la coesione? Quanto sono correlate le operazioni di un elemento software da un punto di vista funzionale

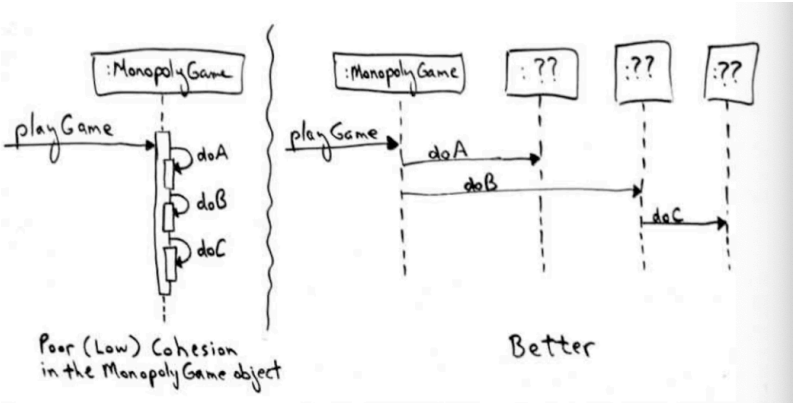
- Come mantenere gli oggetti focalizzati, comprensibili, gestibili e allo stesso momento con poco accopiamento?
- Usare il seguente principio: Assegnare in modo tale che coesione rimanga alta

Livelli di coesione:

- Molto bassa, la classe è sola responsabile di molte cose con funzioni diverse -> Non dovrebbe cadere qui, però è giustificabile
- Bassa, ha 1 responsabilità complessa in una sola area funzionale
- Moderata, 1 responsabilità leggera in poche aree diverse che sono correlate con la classe, ma non tra di loro
- Alta, responsabilità in 1 unica area funzionale e collabora con le altre per svolgere i suoi compiti -> Pochi metodi con funzionalità altamente correlate e focalizzate e fa poco lavoro

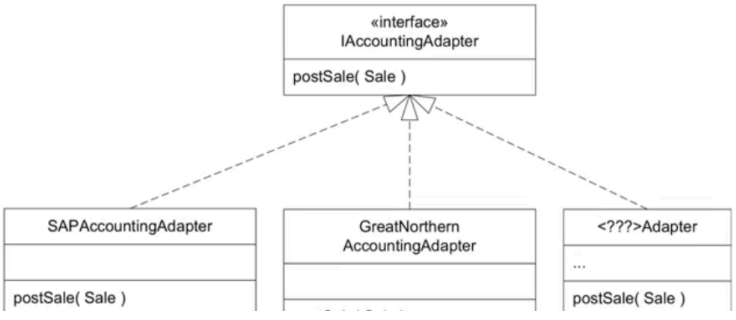
Ci permette più chiarezza e comprensione, riuso siccome ha un accopiamento basso + specializzato e facile da mentenere

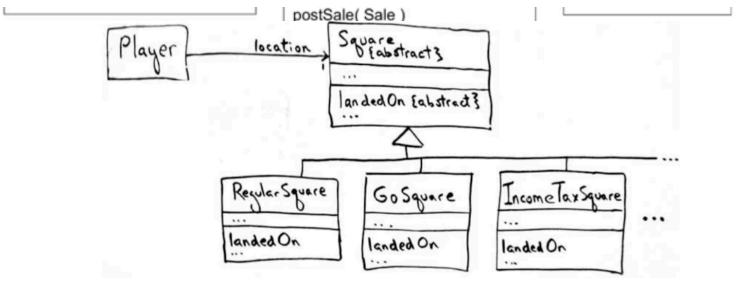
- Controller
  - Qual'è il primo oggetto, oltre all'UI, che riceve e coordina/controlla un operazione di sistema
  - Soluzione: una delle seguenti
    - Rappresenta sistema complessivo, aka un oggetto radice, un punto accesso al software
    - Rappresenta uno scenario di un caso d'uso dove avviene l'operazione di sistema



E' disolito la classe che viene chiamata nello strato del dominio
E' possibile avere più controller
E' un semplice delegatore, non dovrebbe fare altro se non dare ordini senza però esagerare e darli troppe responsabilità (aggiungere più controller, non farli svolgere il lavoro)
Facendo così aumenta il riuso, assicura una seguenzalità

- Polymorphism
  - Problema: come gestire alternative basate sul tipo?
  - Soluzioni: quando alternativi e comportamenti variano a seconda del tipo, assegna responsabilità del comportamento ai tipi per la quale il comportamento varia utilizzando operazione poliformiste
  - Quindi creare sottoclassi e superclassi, questo permette facilità di aggiunta e possibili nuove implementazioni





- Pure fabrication
  - Problema: Un oggetto deve avere delle responsabilità, senza aumentare coesion e coupling, però le soluzioni violano expert
  - Soluzione: Noi creiamo una classe artificiale
  - E' un pò confusionario, praticamente, noi abbiamo una classe generale, e poi questa classe generale verrà messa dentro una classe più specifica.
- Inderaction
  - Problema: dove assegnare una responsabilità per poter evitare l'accoppiamento diretto tra più elementi? Aka disaccoppiare elementi per low coupling
  - Soluzioni: si crea un oggetto intermediario che media tra tanti clienti e servivi
  - Aka, tanti componenti sono interconnessi tra di loro, semplicmenete li connettiamo ad 1 componente.
- Protected variations
  - Problema: COme proteggere oggetti/sistemi/sottoinsiemi affinchè le modifica di X non va verso Y
  - Soluzione: Si identificano i punti dove potrebbero accadere variazioni, e poi si assegnano responsabilità per creare un interfaccia attorno a questi punti (man in the middle again)

Ma cosa sono?

- Principi generali e soluzioni dinamiche ben definite che hanno funzionato in passato
  Composti da:
  - Nome del pattern
  - Problema -> Principio con cui si assegnano le responsabilità
  - Soluzione -> Assegna responsabilità alla classe che può
- E' una coppia problema/soluzione ben nota usata in

            - □ E' una coppia problema/soluzione ben nota usata in passato che può essere applicato in diverse situazioni -> Identificano soluzioni che hanno funzionato in passato, essi **non** affermano nuove idee
            - □ Facilitano la comprensione, memorizzazione e comunicazione siccome tutti la conoscono e comprendono
        - Mentre si disegna un diagramma intereazione vanno prese delle decisione riguardante le responsabilità
    - Design pattern "gang of four"
    - Progettazione guidata dalla responsabilità
        - E' un approccio dove si pensa in termini di responsabilità che bisogna assegnare, ruoli che possono avere e le collaborazione tra oggetti
        - Responsabilità = Ciò che fà o rappresenta un'oggetto
          Aka un contratot o un obbligo di un classificatore
          Ed abbiamo 2 tipi:
            - □ Di fare
                - ◆ Fare qualcosa con se stesso
                - ◆ Delegare ad altri oggetti di fare azioni
                - ◆ Controllare e coordinare oggetti
            - □ Di conoscere
                - ◆ Conoscere i propri dati
                - ◆ Gli oggetti correlati
                - ◆ Ciò che si può derivare oppure calcolare

          Esse vengono assegnate durante la progettazione.
          Esse possono essere trasformati in 1 metodo oppure più metodi nel software
            E quindi i metodi potrebbero collaborare con altri metodi/oggetti
            E da qui è possibile avere una comunità di oggetti con responsabilità che collaborano

          E per progettare le responsabilità si segue:
            - □ Identificare responsabilità iterativamente
            - □ Chiedere a quale oggetto software assegnarlo (nel caso non esiste, crearla)
            - □ Chiedersi come l'oggetto può soddisfare la responsabilità (chiedersi se si ha bisogno di collaborazione, e se non ci sono gli oggetti bisogna crearli)
            - □ Passare alla responsabilità successiva
- Output:
    - Diagrammi UML di iterazioni, classi e package
    - Abbozzi e prototipi dell'interfaccia utente

- Abbozzi e prototipi dell'interfaccia utente
- Modelli delle base di dati

Riassunto:

- Progettazione di interazione tra oggetti
- Identificazione e assegnazione tra gli oggetti
- Il sistema deve essere estendibile, chiaro, mantenibile e riusaible
- I pattern GRASP riassumono alcune scelte per l'assegnazione responsabilità