- ***Lezione 16***

Il diagramma delle attività, è fornito da UML ed ha un potere espressivo molto elevato, poiché permette di rappresentare il flusso di esecuzione in 2 modalità:

- sia sequenziale
- sia concorrente

È una variante degli state diagram, in cui gli stati (i nodi) rappresentano l'esecuzione di azioni e (gli archi) le transizioni sono attivate dal completamento di tali azioni.

Un vantaggio è che questo diagramma può essere anche in assenza del class diagram.

In presenza del class diagram, ogni attività può essere associata ad una o più operazioni appartenenti ad una o più classi

Usato principalmente in fase di OOD per rappresentare il flusso di esecuzione delle operazioni definite nel class diagram. In fase di OOA, viene usato per rappresentare il flusso delle attività nella esecuzione di un caso d'uso (un caso d'uso può essere associato ad uno o più activity diagram).

UML ci fornisce le seguenti primitive:

- si parte da un nodo iniziale che rappresenta l'evento di inizio, che corrisponde all'attivazione di un caso d'uso.
- Le attività da svolgere, che corrispondono ai nodi, ovvero gli action state si identificano pensando ad uno scenario di funzionamento di un caso d'uso
- questi action state sono collegati mediante transition lines (transizioni) che possono essere controllate da guard conditions (condizioni di guardia), il che significa che io una volta terminata un'attività posso andare su quella successiva percorrendo una transizione se e soltanto se la condizione annotata su quella transizione è vera.

Negli Activity Diagram posso rappresentare:

- Flussi concorrenti → un flusso sequenziale viene specificato in più flussi concorrenti. Questo viene realizzato utilizzando delle barre di sincronizzazione, chiamate barre fork - join (fork dove il flusso si splitta, join dove viene modificato)
- Flussi alternativi → vengono modellati con nodi decisionali di branch (che rappresenta flussi alternativi) e merge diamonds (questi flussi alternativi vengono ricombinati) → (rappresentati con dei rombi)
- Eventi esterni non sono generalmente modellati

I Diagrammi di interazione, vengono messi da UML in due modalità:

- Sequence diagram
    - Descrive lo scambio di messaggi tra oggetti in ordine temporale
    - Usato principalmente in fase di OOA
- Collaboration diagram
    - Descrive lo scambio di messaggi tra oggetti mediante relazioni tra gli oggetti stessi
    - Usato principalmente in fase di OOD

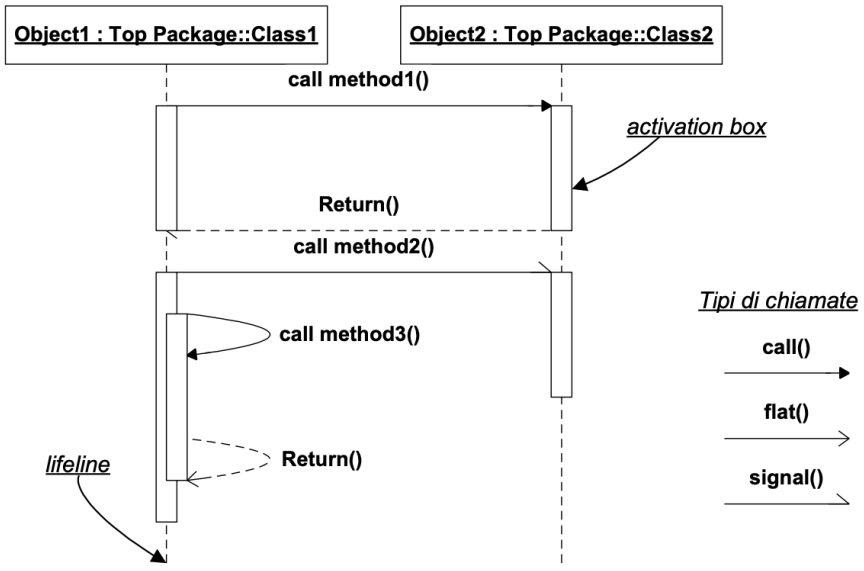
Questi permettono di identificare le operazioni delle classi nel class diagram. Sono rappresentazioni equivalenti e possono essere generati in modo automatico l'uno dall'altro.

Il diagramma di sequenza mostra in modo esplicito le iterazioni tra vari oggetti. Queste iterazioni avvengo attraverso uno scambio di messaggi. Le attività specificate nel diagramma delle attività possono essere mappate come messaggi in un sequence diagram.

Questi messaggi possono essere di due tipi:

- Asincrona → in UML si chiama Signal (segnale). È una richiesta che si basa sul paradigma "Send-no-Replay", l'oggetto mittente continua l'esecuzione dopo aver inviato il messaggio asincrono
- Sincrona → in UML si chiama Call. È una richiesta che si basa sul paradigma "Send-Replay", l'oggetto mittente blocca l'esecuzione dopo aver inviato il messaggio sincrono, in attesa della risposta da parte dell'oggetto destinatario.

Dal punto di vista visuale la notazione è la seguente:



I rettangoli rappresentano gli oggetti che interagiscono. Per ogni oggetto devo dire nome dell'oggetto e classe dal quale quell'oggetto è stato creato.

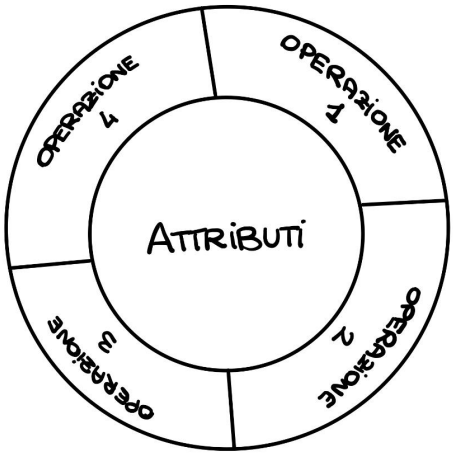
Per ogni oggetto si rappresenta una lifeline (linea tratteggiata) che rappresenta la vita dell'oggetto (scorre dall'alto verso il basso). Qui vengono definiti gli scambi tra messaggi inviati e ricevuti. Queste interazioni tra oggetti vengono rappresentate tramite archi orientati.

Quando l'oggetto durante la sua vita è attivo si usano dei rettangolini vuoti posizionati sulla lifeline, questi si chiamano activation box. [N.B: è una cosa al livello simbolico, infatti UML non ha il concetto di time]

Si parla di Interfaccia Pubblica di Classe. Esiste in Java una tecnica chiamata Information Hiding, la quale si realizza definendo un Interfaccia Pubblica. L'Information Hiding, ha l'idea di implementare le caratteristiche di riusabilità, leggibilità, manutenibilità del codice cercando di far vedere all'esterno soltanto quello che è necessario che gli altri oggetti conoscano. Nel caso del costrutto della classe, quello che ci dice il principio di OOP è che quando si va a definire una classe, gli attributi di questa classe andrebbe nascosta dietro un'interfaccia fornita dalle operazioni. Questi vengono definiti come accessor method che vengono chiamati come getter e setter.

Solitamente una classe viene rappresentata come visto in precedenza, attraverso un rettangolo con 3 "scompartimenti".

Inizialmente esisteva un'altra rappresentazione (che al giorno d'oggi non ne più usata, ma secondo il prof cattura meglio l'essenza dell'Information Hiding):



La "corona" (ovvero il cerchio di operazioni) definisce il concetto di Interfaccia Pubblica di Classe.

Durante la fase di OOA, si determina la signature dell'operazione, che consiste di:

- Nome dell'operazione

- Lista degli argomenti formali
- Tipo di ritorno

Durante la fase di OOD, si definisce l'algoritmo che implementa l'operazione. Una operazione può avere:

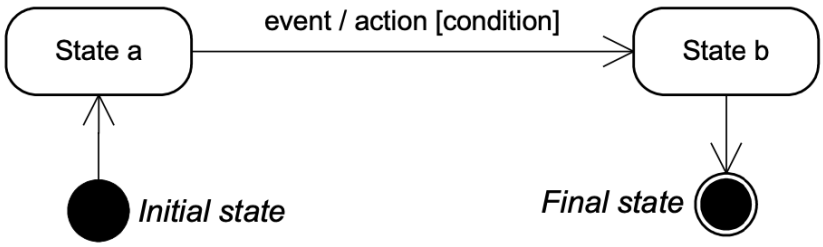
- Instance scope
- Class (static) scope
    - Rappresentata con un carattere $ che precede il nome dell'operazione.
    - Agisce su class object (classi con attributi static)

Per identificare queste operazioni la prima cosa che facciamo è utilizzare tutte le informazioni che ci vengono fornite dal sequence diagram. Si tratta di andare a recuperare per ogni messaggio inviato, il tipo di messaggio, la lista dei parametri, l'eventuale tipo di messaggio di ritorno e aggiungere l'operazione corrispondente alla classe.

Ad ogni classe vanno aggiunte le operazioni che fanno riferimento al criterio CRUD, secondo cui ogni oggetto deve supportare le seguenti operazioni primitive (CRUD operations):

- Create (una nuova istanza)
- Read (lo stato di un oggetto)
- Update (lo stato di un oggetto) • Delete (l'oggetto stesso)

Oltre al modello dei dati e al modello comportamentale serve costruire il modello dinamico. Questo rappresenta il comportamento dinamico degli oggetti di una singola classe, in termini di stati possibili ed eventi e condizioni che originano transizioni di stato (assieme alle eventuali azioni da svolgere a seguito dell'evento verificatosi). Si fa uso del formalismo State Diagrams event / action [condition]



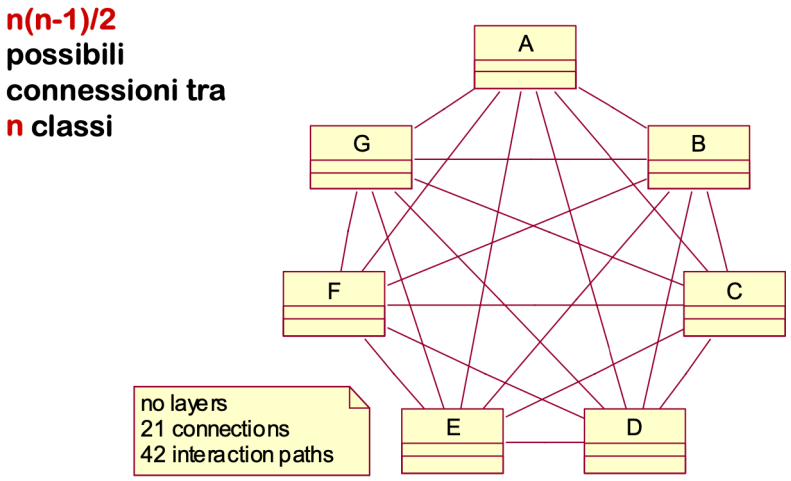
Ogni transizione può essere "etichettata" attraverso → l'evento, la condizione, l'azione

Viene costruito per ogni classe di controllo (per le quali è interessante descrivere il comportamento dinamico)

Usato principalmente per applicazioni scientifiche e real- time (meno frequentemente nello sviluppo di applicazioni gestionali)

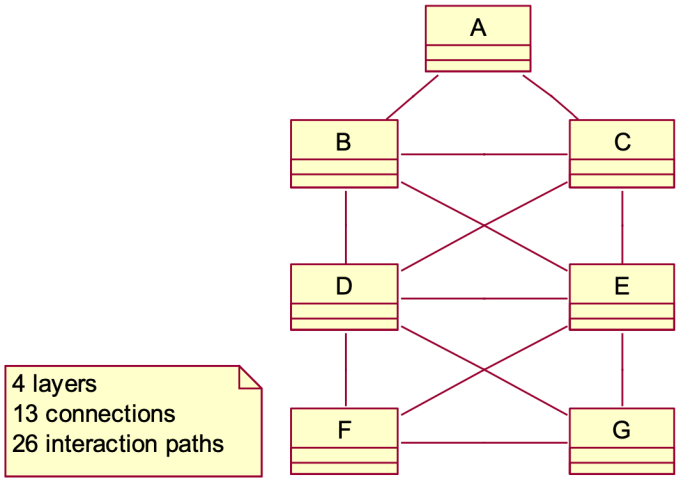
Gestione della complessità nei modelli di OOA. Nella fase di OOA per sistemi software di grandi dimensioni occorre gestire in modo opportuno l'intrinseca complessità dei modelli. Le associazioni tra classi nel modello dei dati formano complesse reti di interconnessione, in cui i cammini di comunicazione crescono in modo esponenziale con l'aggiunta di nuove classi. L'introduzione di gerarchie di classi permette di ridurre tale complessità da esponenziale a polinomiale, grazie all'introduzione di opportuni strati di classi che vincolano la comunicazione tra classi appartenenti allo stesso strato o a strati adiacenti.

- Class diagram non stratificato



Questo è un class diagram interamente connesso, dove ogni classe è associata alle altre classi. Nel nostro caso abbiamo 7 classi e $n\,(\,n - 1\,)\,/\,2$ possibili connessioni, cioè 21.

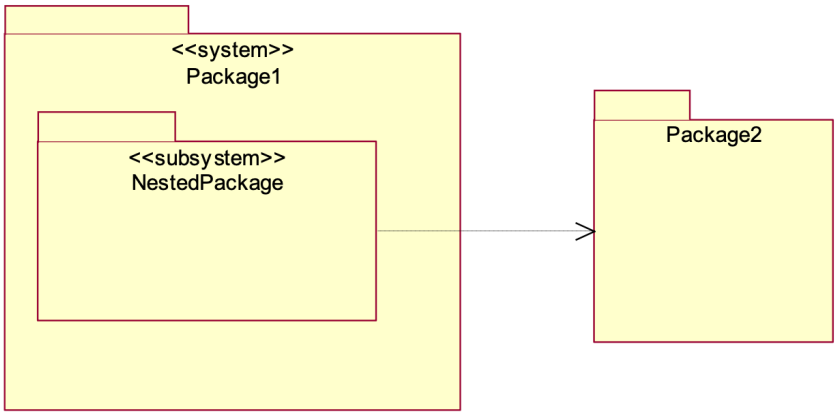
- Class diagram stratificato



Questo è un class diagram stratificato. Qui le classi possono comunicare solo con le classi degli strati adiacenti.

Questi strati vengono definiti attraverso UML Package. L'UML fornisce la nozione di package per rappresentare un gruppo di classi o di altri elementi (ad esempio casi d'uso). I package possono essere annidati (il package esterno ha accesso ad ogni classe contenuta all'interno dei package in esso annidati). Una classe può appartenere ad un solo package, ma può comunicare con classi appartenenti ad altri package. La comunicazione tra classi appartenenti a package differenti viene controllata mediante dichiarazione di visibilità (private, protected, o public) delle classi all'interno dei package.

Per quanto riguarda la rappresentazione grafica della dipendenza tra package avremo:



Si possono specificare due tipi di relazioni tra package:

- Generalization → implica anche dependency
- Dependency → dipendenza di uso, dipendenza di accesso, dipendenza di visibilità

In UML non esiste il concetto di package diagram. I package possono essere creati all'interno di:

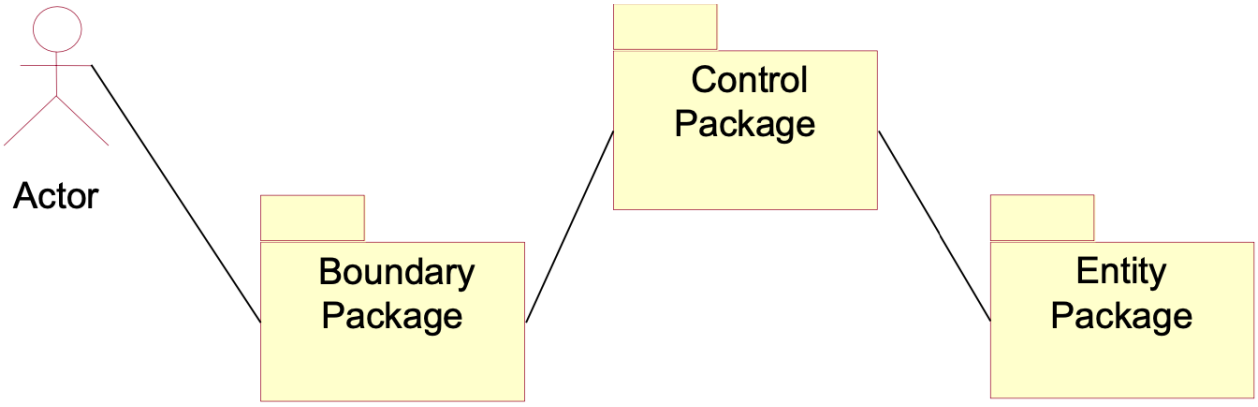
- class diagram
- use case diagram

Per definire quanti sono gli strati e come raggrupparli usiamo l'approccio BCE (Boundary Control Entity):

- Boundary package (BCE) → andremo a inserire le classi di oggetti che gestiscono l'interfaccia tra un attore ed il sistema. Le classi catturano una porzione dello stato del sistema e la presentano all'utente in forma visuale (interfaccia utente)
- Control package (BCE) → raggruppa tutte quelle classi alle quali viene inoltrato l'input dell'utente e devono saper rispondere a questo input. Le classi rappresentano azioni ed attività di uno use case
- Entity package (BCE) → descrive classi i cui oggetti gestiscono l'accesso alle entità fondamentali del sistema. Le classi corrispondono alle strutture dati gestite dal sistema

Questo approccio e simile al paradigma MVC (Model View Controller).

In figura vediamo una gerarchia di package nella quale le richieste degli utenti vanno sugli oggetti boundary, questi interagiscono con gli oggetti di controllo, i quali per poter rispondere possono accedere all'entity package.



N.B: Un oggetto boundary non può comunicare direttamente con un oggetto entity, e viceersa.