PER ALTRI APPUNTI CONSULTARE IL SITO:

https://luigi-v.github.io/Appunti_Universita/

Per altri appunti consultare il sito: *https://luigi-v.github.io/Appunti_Universita/*

By *Vollono Luigi*

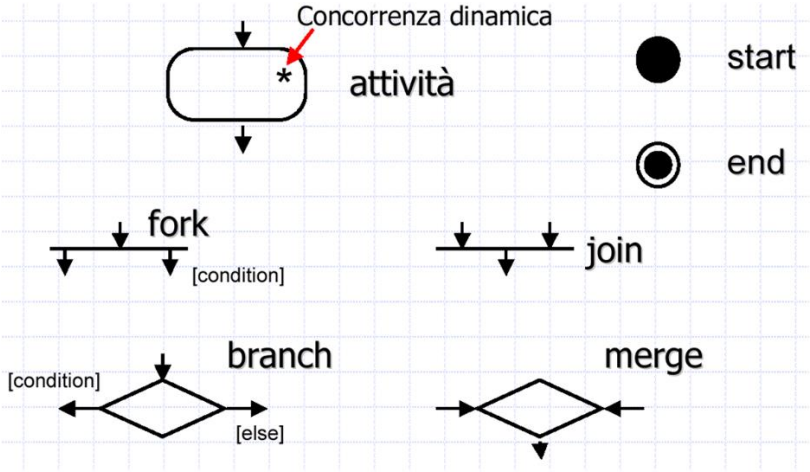
4.5 **Diagrammi delle attività (Activity Diagram)**

Il diagramma delle attività è un diagramma di flusso per rappresentare il flusso da un'attività all'altra. L'attività può essere descritta come un'operazione del sistema. In più permettono di rappresentare processi paralleli e la loro sincronizzazione.

Un Activity Diagram può essere associato:

- A una classe
- All'implementazione di un'operazione
- Ad uno Use Case
- A tutto il sistema
- A processi di business

Elementi Grafici:



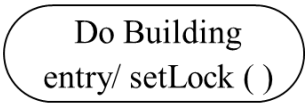
Le attività possono essere gerarchiche

Elementi:

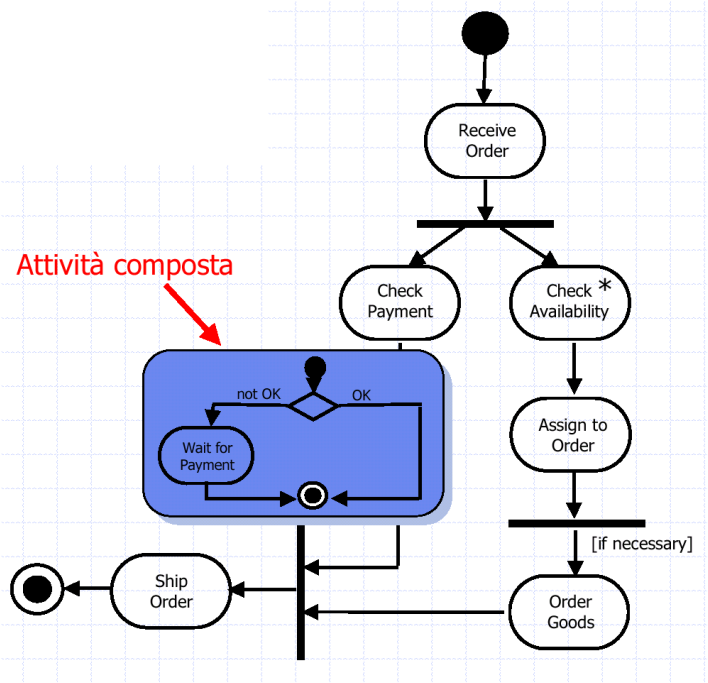
- ***Activity***: una esecuzione non atomica entro uno state machine.
  Una activity è composta da action, elaborazioni atomiche comportanti un cambiamento di stato del sistema o il ritorno di un valore .
- ***Transition***: flusso di controllo tra due action successive.
- ***Guard expression***: espressione booleana (*condition*) che deve essere verificata per attivare una transition.
- ***Branch***: specifica percorsi alternativi in base a espressioni booleane.
  Un branch ha una unica transition in ingresso e due o più transition in uscita.
- ***Synchronization bar***: usata per sincronizzare flussi concorrenti.
  - ***fork***: per splittare un flusso su più transition verso action state concorrenti
  - ***join***: per unificare più transition da più action state concorrenti in una sola

  il numero di fork e di join dovrebbero essere bilanciati
- ***Activity state***: stati non atomici (decomponibili ed interrompibili)
  Un activity state può essere a sua volta rappresentato con un activity diagram.
- ***Action state***: azioni eseguibili atomiche (non possono essere decomposti né interrotti)
  Una action state può essere considerata come un caso particolare di activity state.

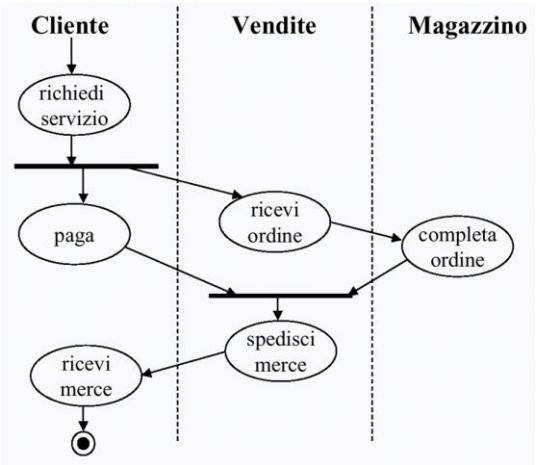
Activity e Action state hanno la stessa rappresentazione grafica, ma un activity state può avere parti addizionali (es. entry ed exit action).



Esempio: Gestione ordini



Swimlanes:



Costrutto grafico rappresentante un insieme partizionato di action/activity. Identificano le responsabilità relative alle diverse operazioni:

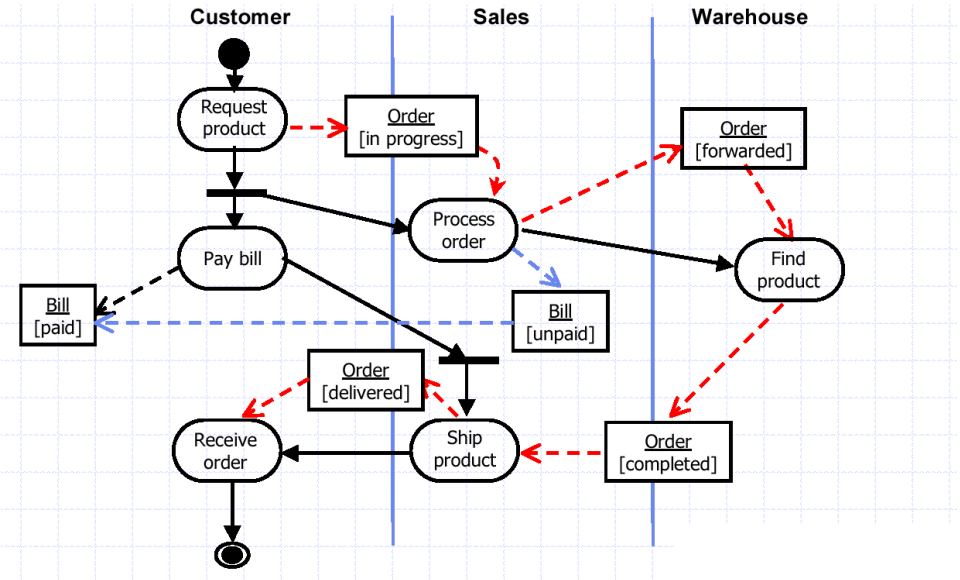
- Parti di un oggetto
- Oggetti diversi

Per ogni oggetto responsabile di action/activity nel diagramma è definito ***swimlane***, identificato da un nome univoco.

Attività e flussi di oggetti (object flow):

Modellano l'utilizzo di object da parte di action/activity state e l'influenza di queste su essi, possono:

- *essere l'output di una action*: la action crea l'object, la freccia della relationship punta l'object
- *essere l'input di una action*: questa usa l'object, la freccia della relationship punta l'action
- *essere manipolati da qualsiasi numero di action*: l'output di una action può essere l'input di un'altra
- *essere presenti più volte nello stesso diagramma*: ogni presenza indica un differente punto della vita dello object



4.1 Class Diagram

Rappresenta un insieme di class, interface, collaboration e le loro relationship.

Definisce la visione statica del sistema, ovvero gli elementi di base del sistema, come:

- ***Classi (di oggetti):*** incapsulamento di struttura dati (stato) e funzioni (operazioni);
- ***Relazioni tra classi***: associazione (uso), generalizzazione (ereditarietà) e aggregazioni (contenimento);

Elementi Class Diagram

Oggetto/istanza di classe:

Rappresenta una qualsiasi cosa, reale o astratta, che abbia un confine definito.

Più in specifico, un oggetto è un qualcosa di identificabile che ricorda il proprio stato e che può rispondere a richieste per operazioni relative al proprio stato.

Gli oggetti interagiscono richiedendo, reciprocamente tra loro, servizi o informazioni (In risposta ad una richiesta, un oggetto può invocare un'operazione che può cambiare il suo stato)

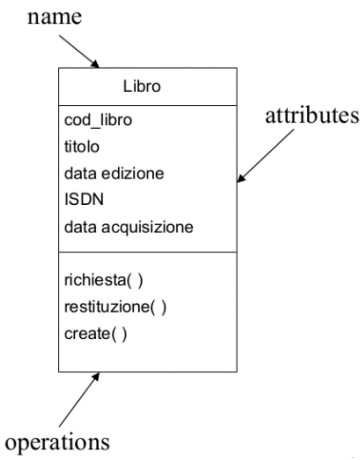
Classe (di oggetti):

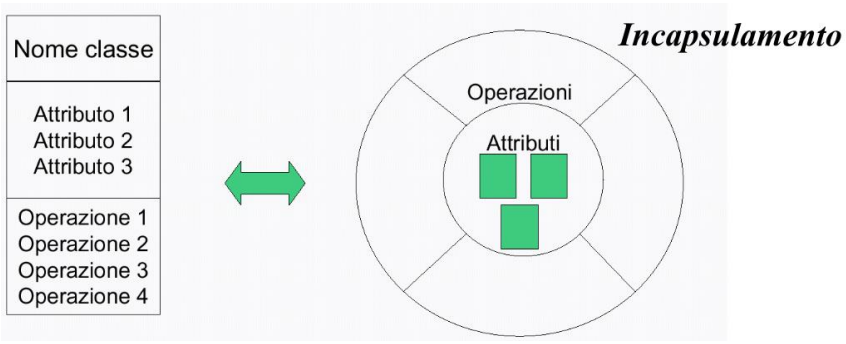
Rappresenta un'astrazione di oggetti che hanno proprietà, come attributi e operazioni.

4.1.1 Classi in UML

In UML una classe è composta da tre parti:

- Nome
- Attributi (stato)
- Metodi o Operazioni (comportamento)

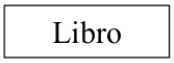




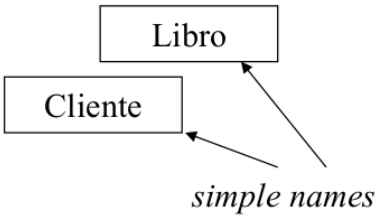
4.1.2 Class - names

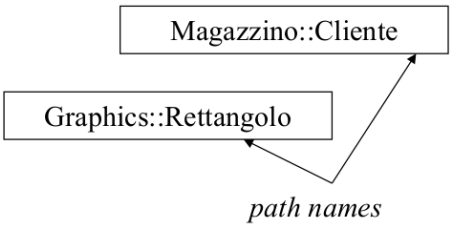
Una classe può essere rappresentata anche usando solo la sezione del nome.

Un nome può essere un:



- ***simple name***, il solo nome della classe;
- ***path name***, il nome della classe preceduto dal nome del package in cui la classe è posta.



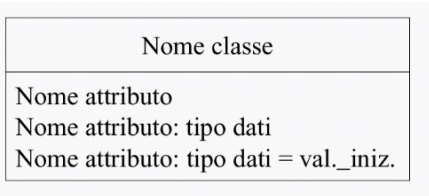


4.1.3 **Class - Attributi**

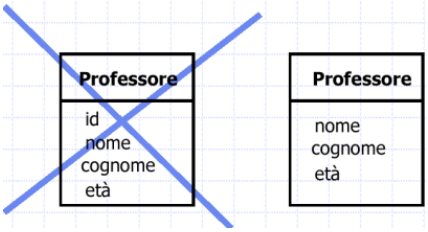
È una proprietà statica di un oggetto, contenente un valore per ogni istanza.

I nomi devono essere unici in ogni classe e possono essere composti e non.

Per ciascun attributo si può specificare il tipo ed un eventuale valore iniziale, il valore non ha identità.

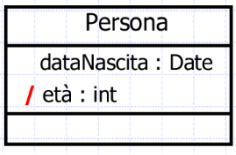


Gli oggetti avranno una loro identità, non bisogna aggiungerla.



Attributi derivati: sono calcolati (in base a valori di altri attributi) e non memorizzati, usati quando i loro valori variano frequentemente ed il suo valore è importante (precisione).

età = f(dataDiNascita, oggi)
area, perimetro = f(vertici)



{età = oggi - dataNascita}

4.1.4 **Class – Metodi/Operazioni**

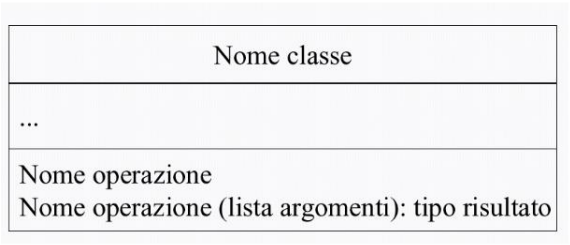
Un'operazione è un'azione che un oggetto esegue su un altro oggetto e che determina una reazione.

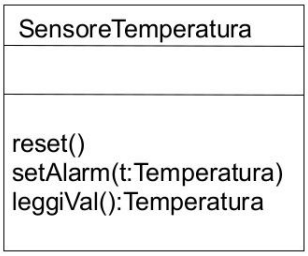
Tipi di operazioni:

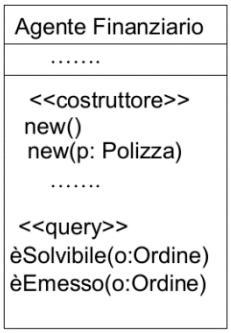
- ***Costruttore:*** crea un nuovo oggetto o inizializza il suo stato;
- ***Distruttore:*** distrugge un oggetto o libera il suo stato;
- ***Selettore (query):*** accedono allo stato dell'oggetto senza alterarlo;
- ***Modificatore***: alterano lo stato di un oggetto.

Per ciascun'operazione si può specificare il solo nome o il suo *signature*, indicando il nome, il tipo, parametri e, in caso di funzione, il tipo ritornato.

Una operazione per la quale esiste lo stesso signature in più classi di una gerarchia è ***polimorphic***.

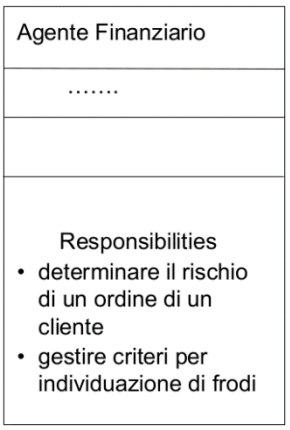






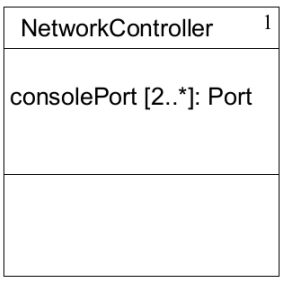
Attributi ed operazioni possono essere mostrati solo parzialmente.

4.1.5 Class – Alcune convenzioni



- ***Responsability***: è un contratto o una obbligazione di una classe, definita dallo stato e comportamento della classe.

- ***Visibilità***: su attributi e operazioni, è possibile specificare tre livelli:
  - + (***public***): qualsiasi altra classe può usare l'attributo o operazione.
  - # (***protected***): qualsiasi classe discendente può visualizzarli
  - - (***private***): solo la classe stessa può usarli



- ***Molteplicità***: è usata per indicare il numero di istanze di una classe, si può applicare agli attributi.

4.1.6 Specifiche sintattiche

La sintassi completa per specificare un **attributi in UML** è:

[visibility] name [[multiplicity]] [: type] [= inital-value] [{property-string}]

dove *property-string* può assumere uno dei seguenti valori:

- ***changeable***: nessuna limitazione per la modifica
- ***addOnly***: una volta creato un valore non può essere né rimosso né modificato
- ***frozen***: il valore non può essere modificato dopo la sua inizializzazione

La sintassi completa per specificare un'**operazione in UML** è:

[visibility] name [(parameter-list)] [:return-type] [{property-string}]

con la *lista dei parametri* avente questa sintassi: **[direction] name: type [=default-value]**

dove *direction* può assumere uno dei seguenti valori:

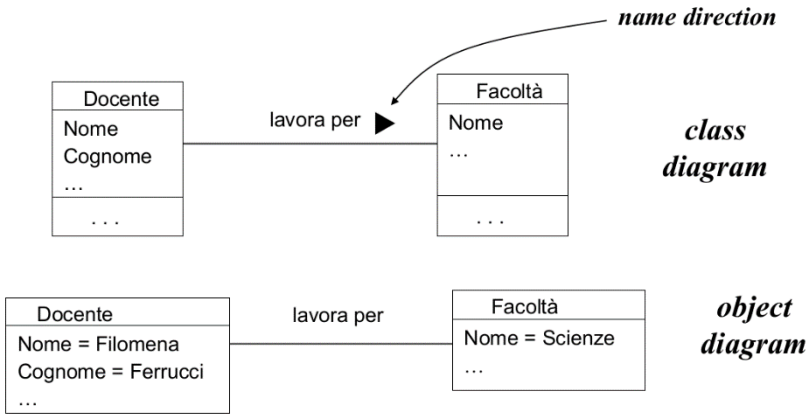
- ***In***: parametro di input
- ***Out***: parametro di output
- ***Inout***: parametro di input/output

4.1.7 **Legami e Associazioni**

Un legame (link) rappresenta una relazione (fisica o concettuale) tra oggetti la cui conoscenza deve essere preservata per un certo periodo di tempo. Es: "*Filomena Ferrucci lavora per Facoltà di Scienze*"

Un' associazione deve avere un nome, di solito un verbo. I link sono istanze delle associazioni:

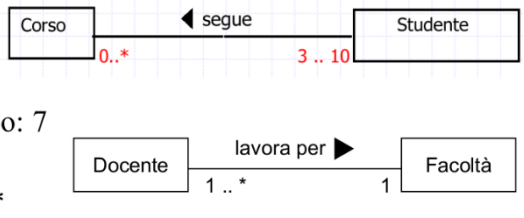
- Un link connette due oggetti
- Un'associazione connette due classi



4.1.8 **Molteplicità delle associazioni**

Utile per rappresentare se l'associazione è obbligatoria oppure no, ed il numero minimo e massimo di oggetti che possono essere relazionati ad un altro oggetto.

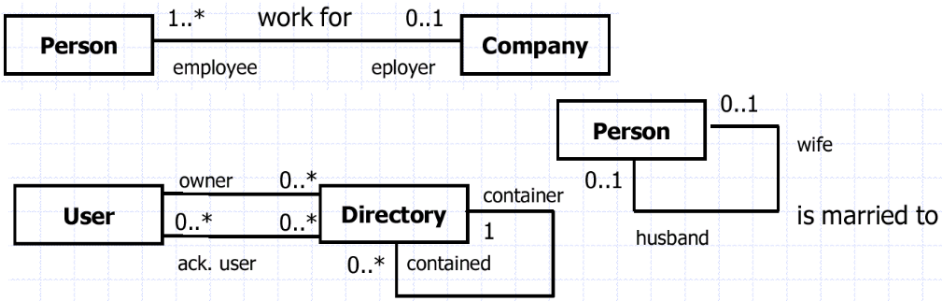
Esattamente uno: 1
Zero o uno: 0..1
Molti: 0..*
Uno o più: 1..*
Un numero specifico: 7
Un intervallo: 4..15
Lista: 0..1, 3..4, 6..*



4.1.9 **Ruoli**

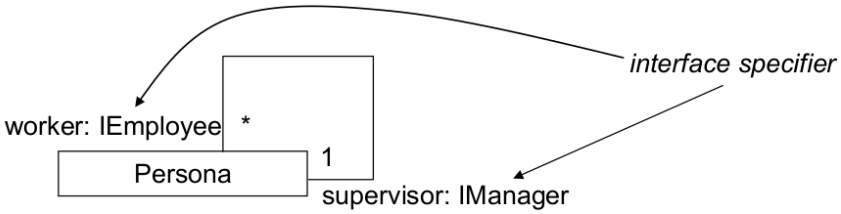
Forniscono una modalità per attraversare relazioni da una classe ad un'altra, i nomi di ruolo possono essere usati in alternativa ai nomi delle associazioni.

I ruoli sono spesso usati per relazioni tra oggetti della stessa classe (associazioni riflessive)

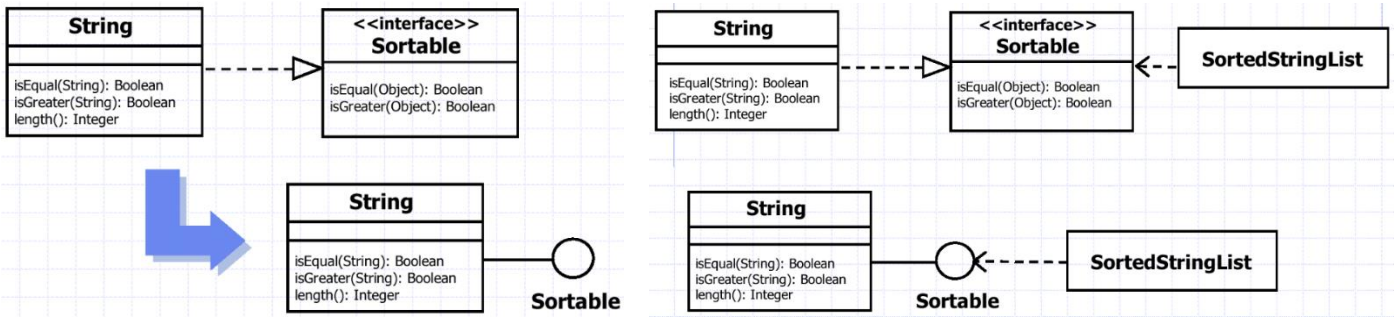


4.1.10 **Interface Specifier**

Una associazione può avere un interface specifier, per specificare quale parte dell'interfaccia di una classe è mostrata da questa nei confronti di un'altra classe della stessa associazione.

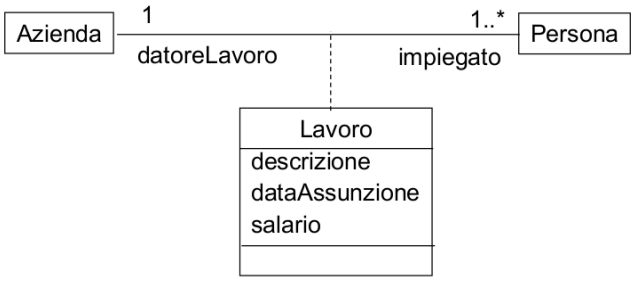


Interfaccia: Specifica il comportamento di una classe senza darne l'implementazione



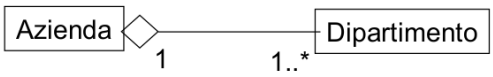
4.1.11 **Classi associate**

Sono utilizzate per modellare proprietà delle associazioni, un attributo di una classe associativa contiene un valore per ogni legame.



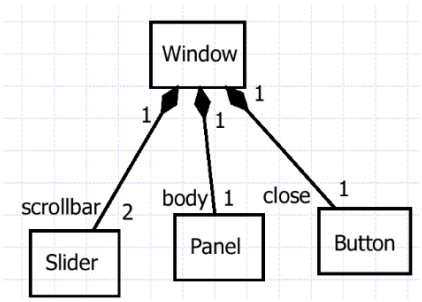
4.1.12 **Aggregazioni**

La relazione di aggregazione è un'associazione speciale che aggrega gli oggetti di una classe componente in un unico oggetto della classe. La si può leggere come "è composto da" in un verso e "è parte di" nell'altro.



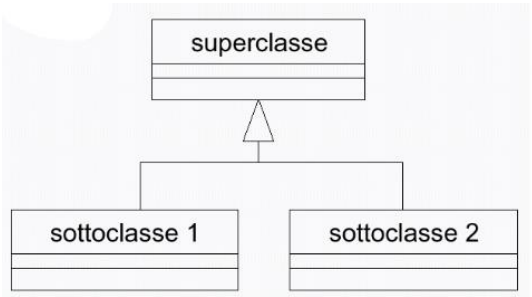
4.1.13 **Composizione**

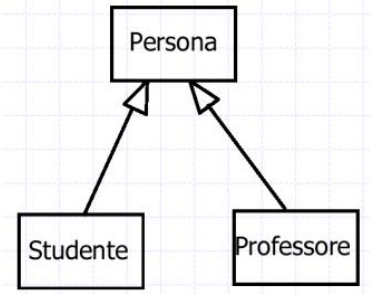
Una relazione di composizione è un'aggregazione forte, le parti componenti non esistono senza il contenitore. Ciascuna parte componente ha la stessa durata di vita del contenitore.



4.1.14 **Generalizzazione**

La relazione di generalizzazione rappresenta una appartenenza delle classi. Può essere letta come “è un tipo di” (verso di generalizzazione), “può essere un” (verso di specializzazione).



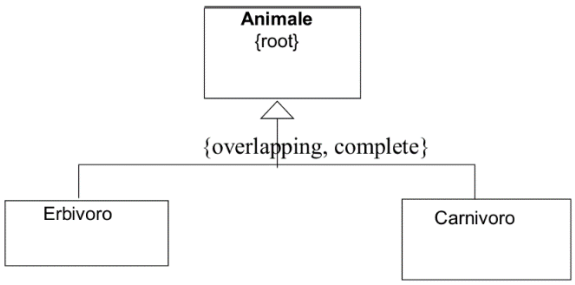


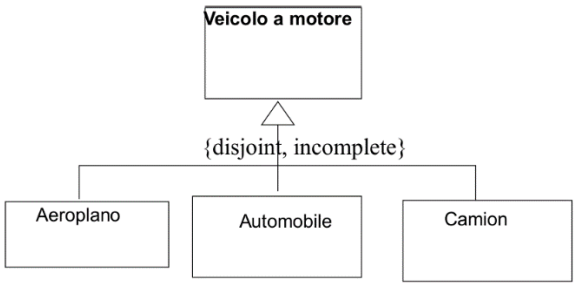
Per specificare che una class non può avere discendenti si usa la proprietà ***leaf*** sotto il name della classe.

Per specificare che una classe non può avere antenati si usa la proprietà ***root*** sotto il name della classe.

disjoint: oggetti del genitore possono avere non più di un figlio come tipo.

overlapping: oggetti del genitore possono avere più di un figlio come tipo.

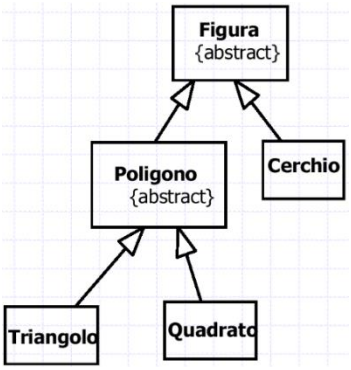




4.1.15 **Classi astratte**

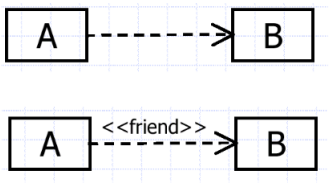
Una classe astratta definisce un comportamento “generico”. Definisce e può implementare parzialmente il comportamento, ma molto della classe è lasciato indefinito e non implementato.

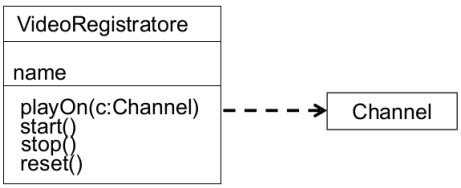
Per indicare che una operazione è astratta il suo nome sarà scritto in corsivo.



4.1.16 **Dependency**

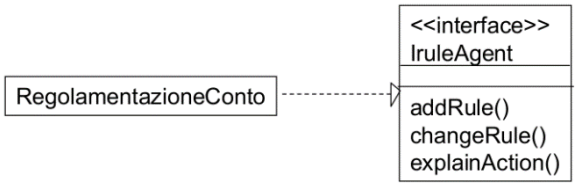
Relazione semantica in cui un cambiamento sulla classe indipendente può influenzare la semantica della classe dipendente.





4.1.17 **Realization**

Una relazione tra classi specifica un contratto che l'altra garantisce di compiere. La freccia punta alla classe che definisce il contratto.



4.1.18 **Costruzione Class Diagram**

Obiettivo è identificare gli elementi del modello a oggetti e come sono in relazione tra loro. Ogni diagramma deve avere lo scopo di:

- mostrare le classi e gli oggetti che partecipano in una singola collaborazione
- mostrare una tassonomia di generalizzazione
- mostrare la suddivisione delle partizioni logiche (packages)

L'ordine non è rigido e le iterazioni sono numerose:

1. ***Identificare le classi di oggetti***:
   Conoscere il problema generale, cercare oggetti fisici, dispositivi, eventi da ricordare, ruoli, locazioni, unità organizzative o altri sistemi. Considerare candidati aventi confini ben definiti e identità distinte, scartando quelli ridondanti o vaghi, e che rappresentano singoli oggetti o operazioni su oggetti.

Classi selezionate
- Conto
- Bancomat
- Banca
- Scheda
- Cassiere
- Terminale del cassiere
- Cliente
- Transazione

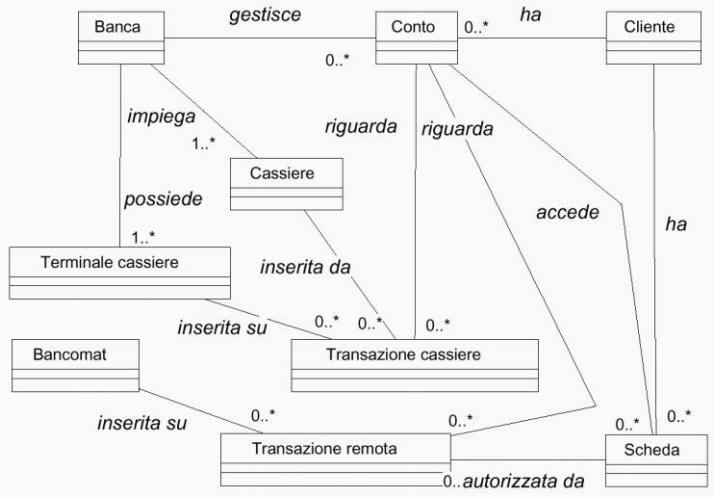
- Classi scartate
  - Vaghe
    - » Sistema, Provvedimento di sicurezza
    - » Provvedimento per la registrazione
    - » Rete bancaria
  - Singoli oggetti
    - » consorzio
  - Attributi
    - » Dati del conto, Ricevuta, Contante
    - » Dati della transazione
  - Irrilevanti
    - » Costo
  - Realizzazione
    - » Identificazione della transazione
    - » Accesso, Software, Linea di comunicazione

2. ***Preparare un dizionario dei dati***:
   Definire cosa la classe rappresenta nel contesto del problema.
   Specificare quali sono le responsabilità della classe nel sistema.

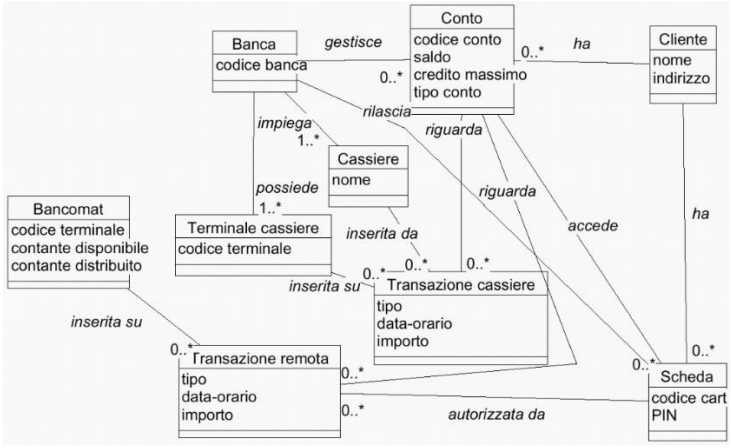
Conto
- *Rappresenta un singolo conto di un cliente in una banca. Tutti gli accessi e le modifiche al conto della banca devono avvenire attraverso questa classe*

Transazione Remota
- *Una richiesta integrale di operazioni sul conto effettuata dal cliente attraverso un Bancomat*

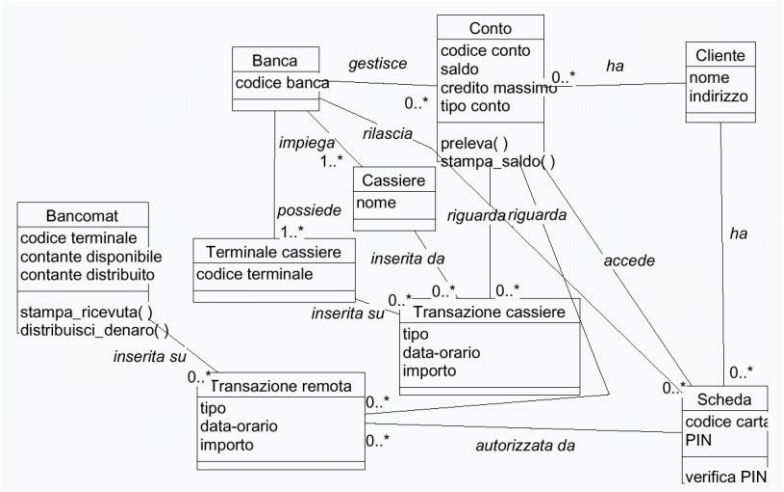
3. ***Identificare le associazioni (incluse le aggregazioni) tra classi***:
   Cercare le dipendenze tra classi, considerando le espressioni verbali nella descrizione del problema. Aggiungere i nomi alle associazioni o ruoli delle classi associate. Scartare le associazioni non rilevanti o troppo legate alla realizzazione, e quelle derivabili da altre associazioni.



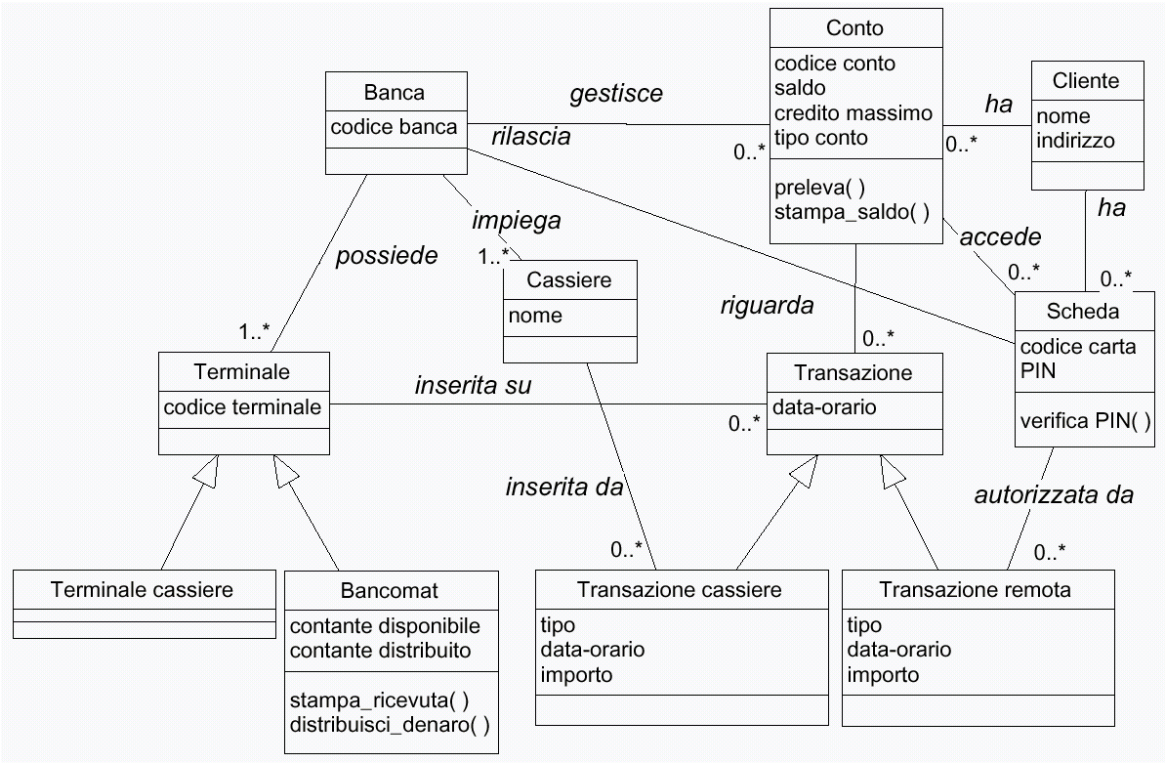
4. ***Identificare gli attributi delle classi e dei legami***:
   In fase di analisi omettere attributi derivati e che descrivono stati interni alla classe.



5. ***Identificare le operazioni***:
   Costruttore, selettore, modificatore, distruttore, consultare gli scenari che descrivono i casi uso.
   Considerare le operazioni che modificano lo stato degli oggetti.



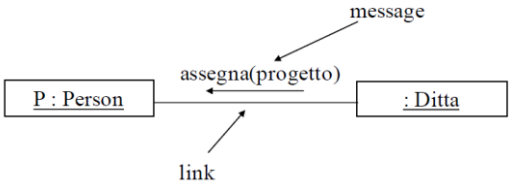
6. ***Riorganizzare usando l'ereditarietà***:
   Vedere se è possibile suddividere una classe in sottoclassi, e identificare una superclasse che astrae attributi e operazioni comuni.



4.3 Sequence Diagrams (Interaction Diagram)

Descrivono le sequenze di azioni e le interazioni tra le componenti, servono a dettagliare gli ***use case***, definire le interfacce del sottosistema e per trovare tutti gli oggetti partecipanti al sistema, evidenziando la sequenza temporale delle azioni. Sono utili per evidenziare la distribuzione del controllo nel sistema ("chi" fa "che cosa" …).

Una ***interazione*** è un comportamento che comprende un insieme di messaggi scambiati tra un insieme di oggetti per ottenere un risultato, avviene tra oggetti tra cui esiste un link (istanza di un'associazione).



Gli oggetti collaborano scambiandosi ***messaggi*** che specificano la comunicazione tra oggetti che trasmettono informazioni.

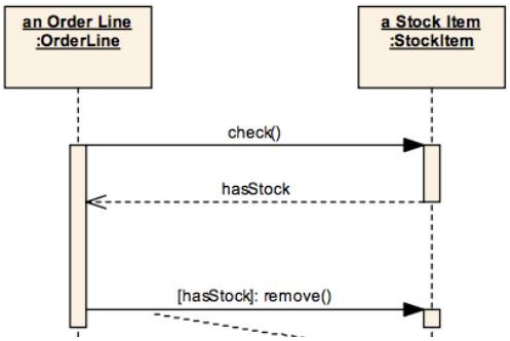
Per l'invio di un messaggio è necessario: ricevente, messaggio e eventuali informazioni aggiuntive.

4.3.1 Componenti Sequence Diagrams

- ***Linee verticali***: rappresentano le attività svolte dagli oggetti;
- ***Linee orizzontali***: rappresentano sequenze dei messaggi scambiati tra oggetti;
- ***Sequence Diagrams***: possono corrispondere ad uno scenario o ad un intero caso d'uso (si possono annotare vincoli temporali);
- ***Oggetti***: istanza di una classe, sono disposti orizzontalmente. Sintassi: ***nomeOggetto : NomeClasse;***

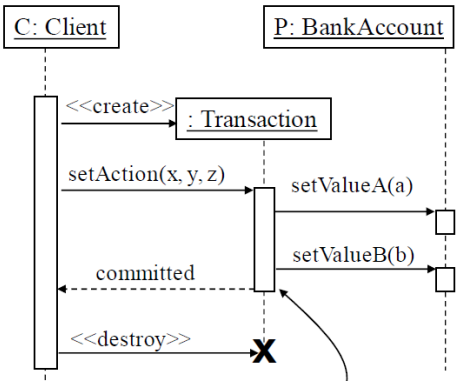
- ***Life-time di un oggetto***: linea tratteggiata verticalmente all'oggetto.
- ***Flusso del Tempo***: è descritto verticalmente;
- ***Life-time di un metodo***: rappresentato da un rettangolo che collega la freccia di invocazione con la freccia di ritorno, rappresentata con una freccia tratteggiata (ritorno opzionale, se si omette, la fine è decretata dalla fine del life-time).

- ***Messaggi sincroni***: si disegna con una *freccia chiusa*, etichettata col nome del metodo. Il chiamante attende la terminazione del metodo prima di proseguire.
- ***Messaggi asincroni***: usati per descrivere messaggi concorrenti, si disegna con una *freccia aperta*, etichettata col nome del metodo. Il chiamante non attende la terminazione, ma prosegue subito dopo l'invocazione.

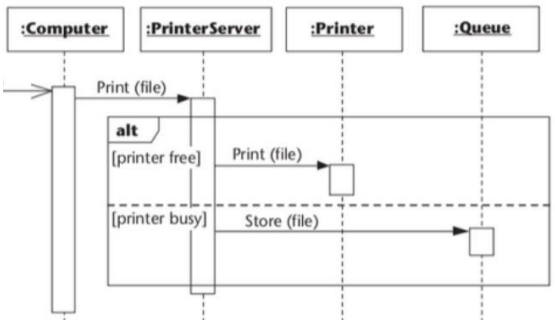


- ***Condizione di un metodo***: invocato solo se la condizione è verificata, posizionata sulla freccia del metodo. Sintassi: ***[cond]:nomeMetodo()***.
- ***Messaggi iterativi***: sintassi: ****messaggio()***
- ***Messaggi condizionati***: sintassi ***[cond]messaggio()***

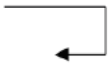
- ***Costruzione nuovo oggetto***: corrisponde all'allocazione dinamica. Etichettata con ***new*** o ***create***.
- ***Distruzione oggetto***: corrisponde alla deallocazione dinamica. Etichettata con ***X*** posta alla fine dell'attività dell'oggetto.
- ***Box di attività***: indica il periodo di tempo durante il quale un oggetto sta eseguendo una action.



- ***Iterazioni (loop)***: esecuzione ciclica di più messaggi, si disegna raggruppando con un blocco i messaggi su cui si vuole iterare. Si può aggiungere la condizione di iterazione, rappresentata tra parentesi quadre.
- ***If-then-else (Alt)***: la condizione si indica in cima, se ci sono else si usa una linea tratteggiata per separare la zona then.
- ***If-then (Opt)***: sequenza che viene eseguita solo se la condizione è verificata.



- ***Auto-chiamata***: oggetto che invoca un proprio metodo. Rappresentato con una freccia circolare.



4.3.2 Euristiche Sequence Diagrams

Le colonne rappresentano gli oggetti che partecipano al caso d'uso.

- ***1 colonna***: rappresenta l'attore che inizia lo *use case*;
- ***2 colonna***: rappresenta l'oggetto *boundary* con cui l'attore interagisce per rappresentare lo *use case*;
- ***3 colonna***: rappresenta l'oggetto *control* che gestisce il resto dello *use case*;
- ***Dalla 4 colonna***: vengono rappresentati gli oggetti *entity*.

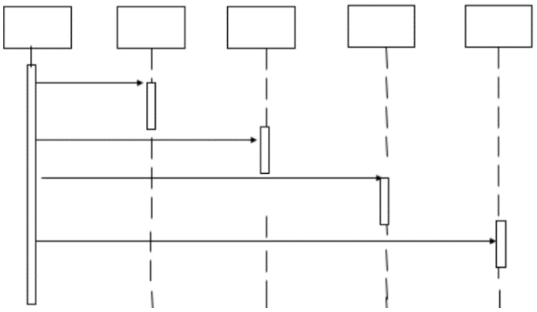
Gli oggetti control creano oggetti boundary e possono interagire con altri control. Gli oggetti *entity* non devono mai chiamare oggetti *control* o *boundary*.

Al top del diagramma si trovano oggetti esistenti dall'inizio.

4.3.3 Fork Diagram

Gran parte del comportamento dinamico è inserito in un singolo oggetto, generalmente l'oggetto di controllo.

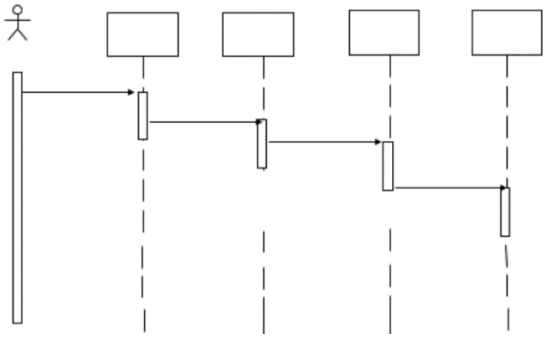
Conosce tutti gli altri oggetti e li usa spesso per domande e per comandi diretti.



4.3.4 Stair Diagram

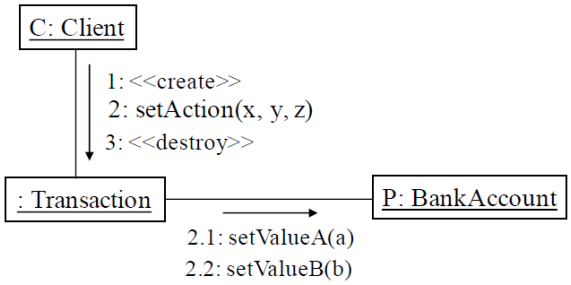
Il comportamento dinamico è distribuito. Ogni oggetto delega alcune responsabilità ad altri oggetti.

Ogni oggetto conosce solo alcuni degli altri oggetti e sa quali oggetti possono aiutare con un comportamento specifico.

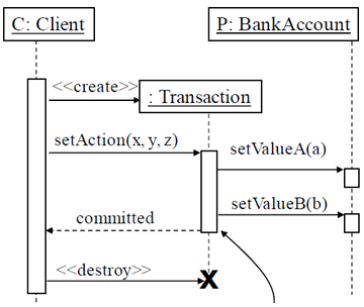


4.3.5 Collaboration Diagram

La sequenza dei messaggi è meno evidente che nel diagramma di sequenza, mentre sono più evidenti i legami tra gli oggetti. Adatti per concorrenza e thread, invocazioni innestate.



→



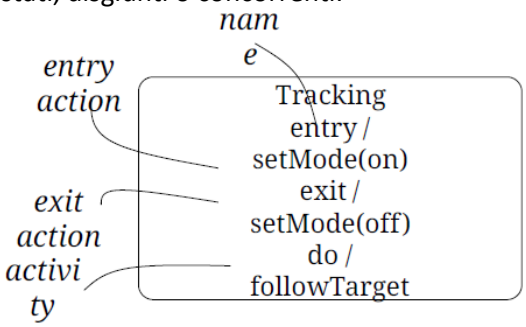
4.4 **Diagramma a stati (state chart)**

Specifica il ciclo di vita di un oggetto (istanza di classe, caso d'uso, sistema). Rappresentano il comportamento dei singoli oggetti in termini di:

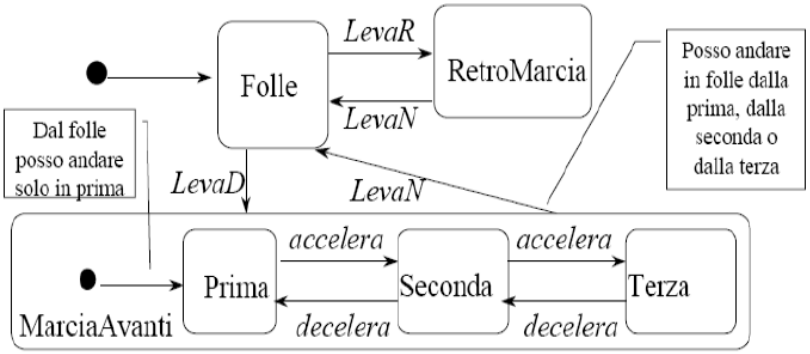
- Eventi a cui gli oggetti (la classe) sono sensibili
- Azioni prodotte
- Transizioni di stato, identificazione degli stati interni degli oggetti

Questo diagramma viene definito per una ***classe***, ed intende descrivere la sua evoluzione. Elementi in esso:

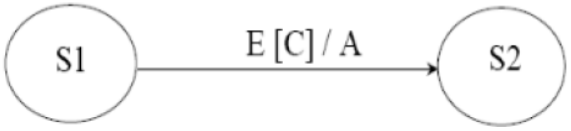
- **Stato**: rappresenta una situazione in cui un oggetto ha un insieme di proprietà considerate stabili.
  Lo stato può avere degli attributi:
  - *Nome*: stringa che specifica un evento
  - *Entry/Exit actions*: eseguite all'ingresso/uscita dallo stato (non interrompibile, durata istantanea)
  - *Transazioni interne*: non causano un cambiamento di stato (Stato finale = Stato iniziale)
  - *Attività*: interrompibile, durata significativa
  - *Sottostati*: struttura innestata di stati, disgiunti o concorrenti.

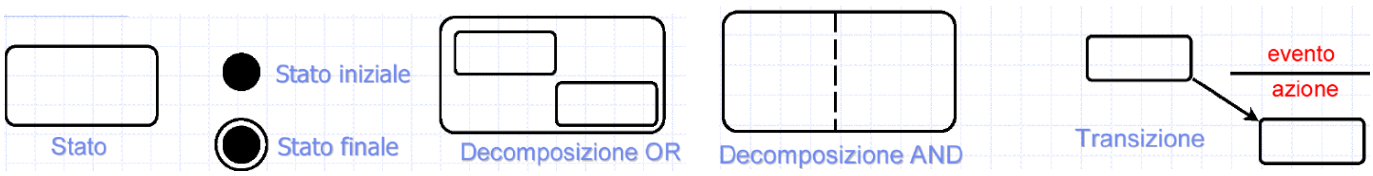


  Stato composto: stato che ha un nome, e che contiene a sua volta un diagramma. Esiste uno stato iniziale del macro-stato. I sottostati ***ereditano*** le transazioni in uscita del macro-stato.

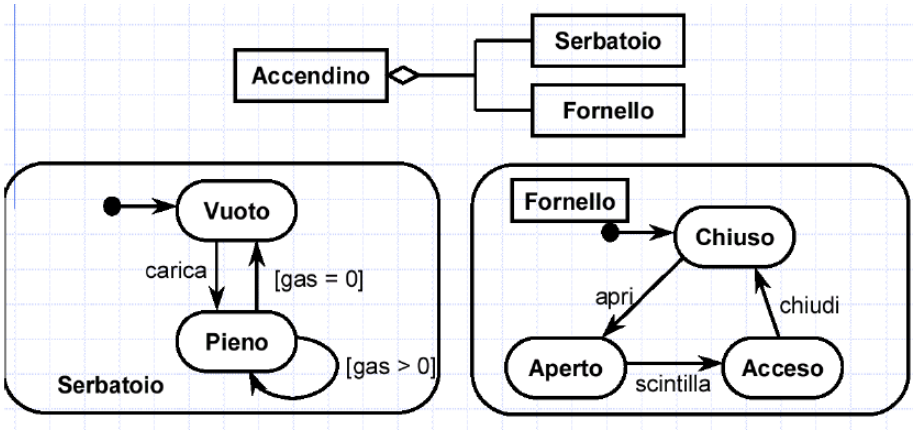


- **Transizione**: modella un cambiamento di stato ed è denotato: **Evento [Condizione]/ Azione**
  Ogni transizione connette due stati, e come se un evento fosse un input e l'azione un output.
  La condizione è detta anche "***Guardia***".
  L'evento è quasi sempre presente, mentre condizione e azione sono opzionali.
  La transazione è un cambiamento da uno stato iniziale ad uno finale
  Esempio:
  - Se l'oggetto si trova nello stato S1, e
  - Se si verifica l'evento E, e
  - Se la Condizione C è Verificata
  - Allora viene eseguita lazione A e l'oggetto panna nello stato S2

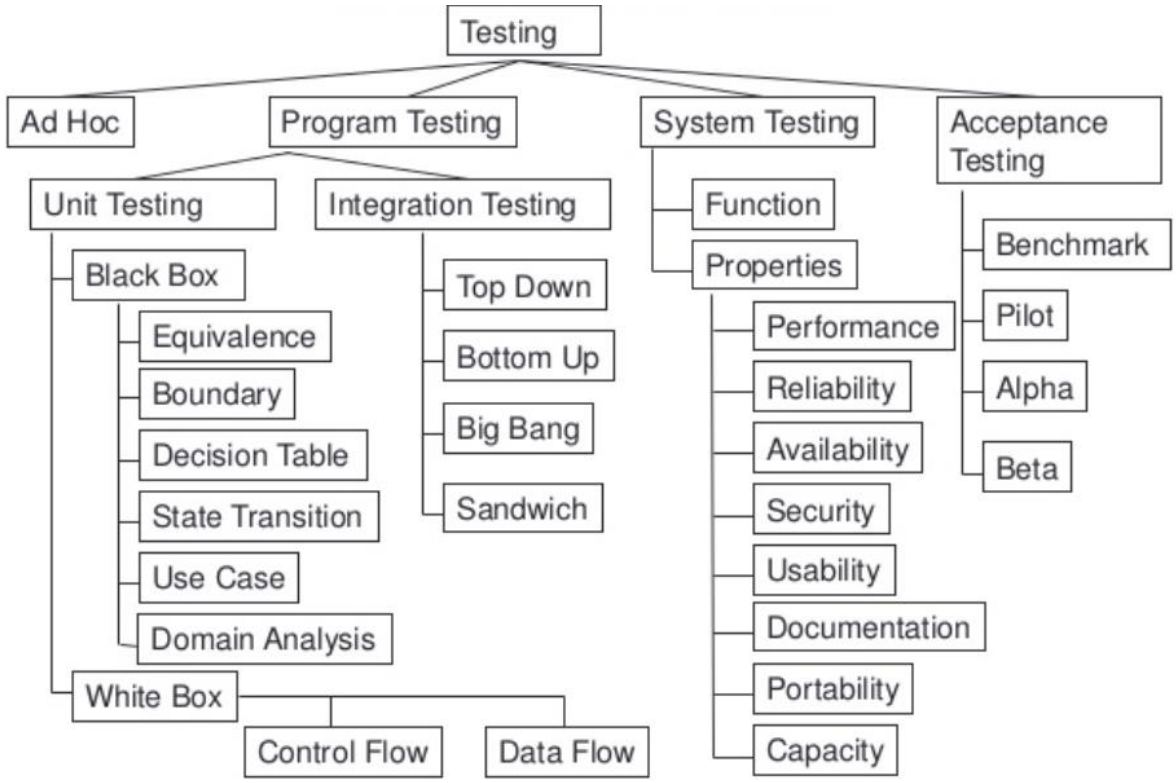




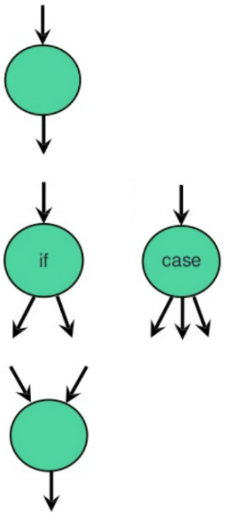
- **Decomposizione**: più stati sono eseguiti in parallelo entro lo stato che li racchiude. Quando avviene una decomposizione, il flusso subisce un fork per ciascun sottostato, alla fine si ricompone in un unico flusso con un join. Se un sottostato raggiunge la fine prima dell'altro, il controllo aspetta lo stato finale dell'altro.



GERARCHIA DI TESTING



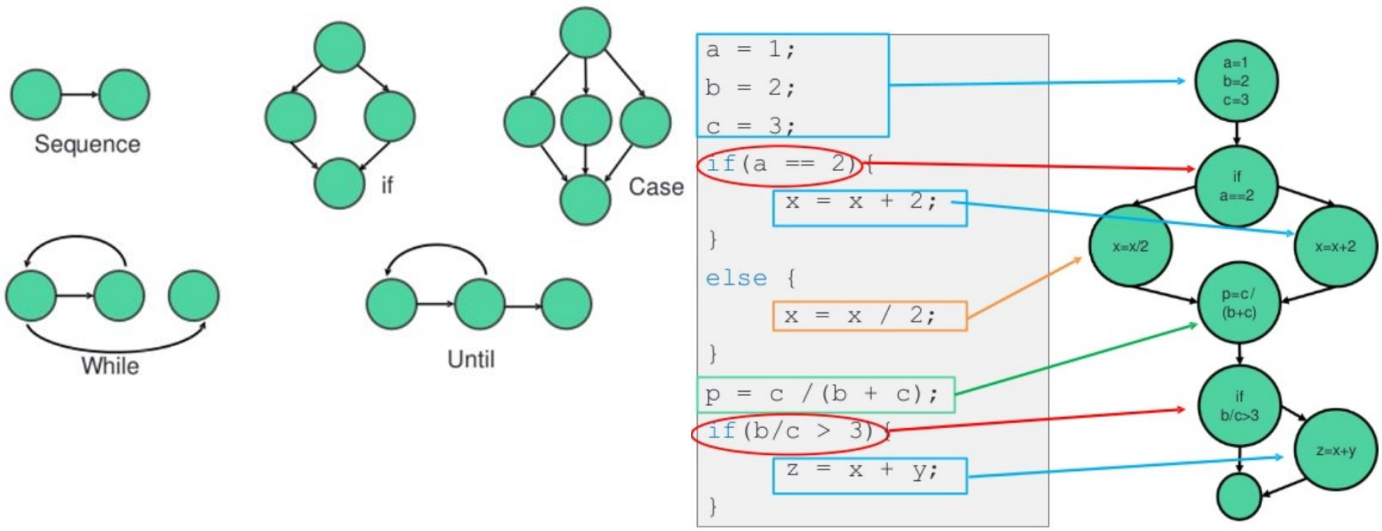
Control Flow Graphs:



- ***Process Blocks***: sequenza di istruzioni di programma che si eseguono in sequenza:

- ***Decision Point***: punto del modulo in cui il flusso può cambiare. La maggior parte sono binari (if-else), ci possono essere anche casi multipli (case):

- ***Junction point:*** punto in cui i flussi di controllo si uniscono:

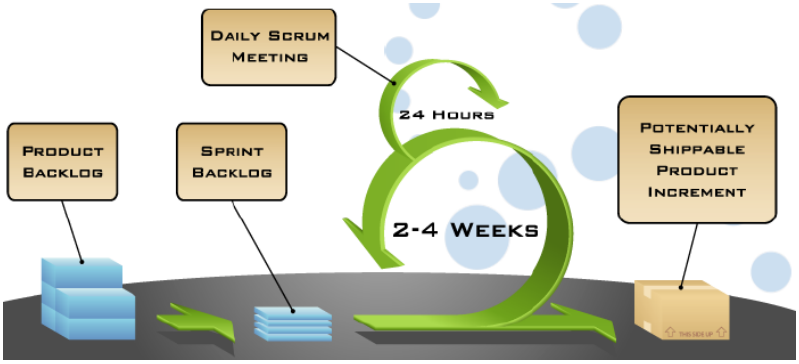


SCRUM

Scrum è un "*framework agile*", iterativo ed incrementale, che ci consente di concentrarci sulla consegna del massimo valore aziendale nel minor tempo possibile.

Ogni due settimane o un mese chiunque può vedere il software funzionante e decidere di rilasciarlo così com'è o continuare a migliorarlo per un altro sprint.

- *Il prodotto avanza in una serie di "sprint" di un mese*
- *I requisiti vengono acquisiti come elementi in un elenco di "Product backlog"*
- *La durata tipica è di 2-4 settimane o un mese al massimo*
- *Il prodotto è progettato, codificato e testato durante lo sprint*
- *Durante lo sprint i requisiti non possono cambiare*



RUOLI:

- ***Product owner***, definiscono le caratteristiche del prodotto ed il suo rilascio.
- ***ScrumMaster***, responsabile della gestione del progetto, rimuove gli impedimenti e protegge il team da interferenze
- ***Team***, si auto-organizzano per determinare il modo migliore per fornire le funzionalità con la massima priorità.

ARTEFATTI:

- ***Team Capacity*** è calcolata secondo la disponibilità delle persone in quello sprint.
- ***Team Velocity*** è l'unità di misura che consente di tenere traccia di quanto lavoro può svolgere il team durante uno sprint.
- ***Focus Factor (F.F)*** è la capacità delle squadre di rimanere concentrati sugli obiettivi dello sprint senza altre distrazioni.
- ***User Stories*** sono artefatti di requisiti molto sottili e di alto livello.
- ***Product Backlog***, rappresenta cosa deve essere fatto, è una lista ordinata dei requisiti relativi al prodotto. Contiene Item a cui viene assegnata una priorità.
- ***Sprint Backlog***, è la lista del lavoro che il team di sviluppo deve effettuare nel corso dello sprint successivo, generata selezionando un insieme di Product Backlog
- ***Burndown Charts***, è una rappresentazione grafica del lavoro da fare su un progetto nel tempo. Di solito il lavoro rimanente (o backlog) è indicato sull'asse verticale e il tempo sull'asse orizzontale, utile per prevedere quando avverrà il completamento del lavoro.

EVENTI:

- ***Sprint planning***, posto all'inizio di ogni sprint dove viene pianificato il lavoro da svolgere, viene creato lo *Sprint Backlog*.
- ***Daily scrum meeting***, ogni giorno durante lo Sprint, viene tenuta una riunione del team di progetto. Ogni membro risponde a:
  - Che cosa è stato fatto dopo l'ultima riunione?
  - Che cosa si farà prima della prossima riunione?
  - Quali sono gli impedimenti / ostacoli incontrati?
- ***Sprint review***, il team presenta ciò che ha realizzato durante lo sprint, se necessario vengono fatte modifiche.
- ***Sprint retrospective***, il Team ispeziona il suo lavoro e crea un piano di miglioramento da attuare durante il prossimo Sprint.