Analisi e Sviluppo di una applicazione con OOP

Java e la progettazione ad oggetti

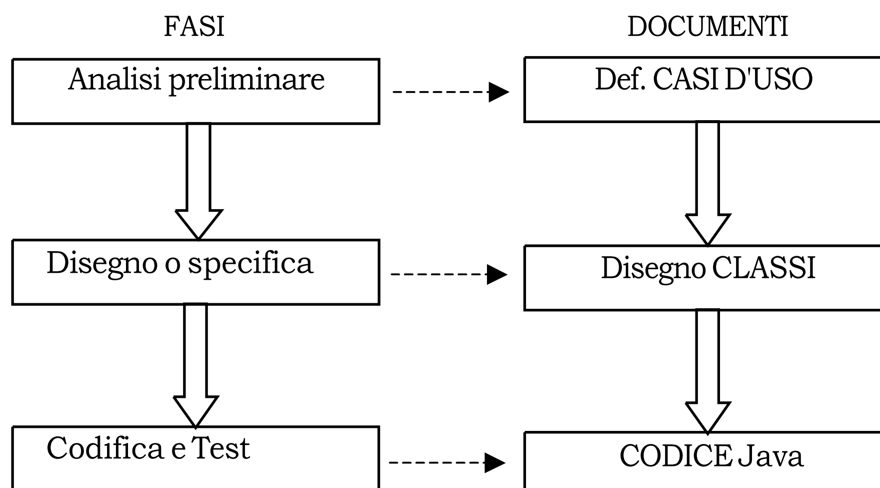
In generale per passare dalla programmazione imperativa-procedurale del C alla pro grammazione ad oggetti in Java è necessario “ricostruire” un quadro di riferimento concettuale “nuovo”.
In C si ragiona in termini di Algoritmi e di Strutture Dati. Nella programmazione ad oggetti si potrebbe dire che un Programma è una Costruzione “tipo Lego” e i mattoni che la compongono sono gli “Oggetti”.
Un oggetto (più esattamente una classe) è un “contenitore” costituito dai Dati che lo caratterizzano e dalle Operazioni che intervengono su quei dati. In termini più precisi una classe è un’entità unica costituita da tre parti essenziali:

- **NOME** (che individua la classe univocamente)
- **ATTRIBUTI** o **CAMPI** (Dati che caratterizzano la classe)
- **METODI** (Servizi forniti dalla classe o Operazioni che intervengono sugli attributi).

Linguaggio e fasi di progettazione

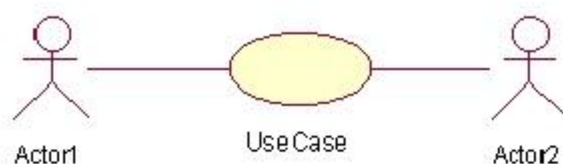
In precedenza si sono utilizzate alcune classi presenti nel "jdk" di java quali String, StringBuffer, Integer o Double e si è visto il manuale della Documentazione di Java e la sua organizzazione in "Packages" che sono Raggruppamenti di classi affini.

Ora si cercheranno di definire alcuni criteri necessari per progettare un programma con una metodologia orientata agli oggetti Object Oriented (OO).
Di norma si parte da una Situazione Problematica (situazione più o meno definita dalle esigenze dell'utente finale o testo del problema) e si affrontano diverse fasi che, in sequenza,
sono: **l'Analisi**, il **Disegno** e la vera e propria **Programmazione**.



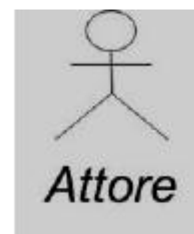
Nella fase di analisi **OOA (Object Oriented Analysis)** si analizza la situazione problematica e si definiscono i "**casi d'uso**" finali, i requisiti, e le funzionalità richieste dall'utente. Occorre porsi dal punto di vista dell'utilizzatore finale e usare le conoscenze di sistema che di norma sono spesso esterne all'ambito dell'informatica .
Ai casi d'uso individuati si assegna un nome e si schematizza la situazione con il diagramma dei casi d'uso, eventualmente accompagnato da un testo esplicativo.

Gli Use Case sono collezioni di scenari che riguardano l'utilizzo del sistema in cui ogni scenario descrive una sequenza di eventi.
La sequenza di eventi descritta da uno Use Case viene iniziata da una **persona**, o da un **altro sistema** o da un **pezzo di hardware** o ancora dal **passare del tempo**. Le entità che iniziano la sequenza di eventi sono definiti **Actors**. Il risultato della sequenza deve portare a qualcosa di utile all'actor che ha iniziato la sequenza o, anche, ad un differente actor.



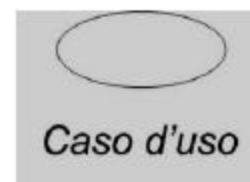
Un actor inizia la sequenza di un particolare use case, ed un actor (possibilmente lo stesso che ha iniziato, ma non necessariamente) riceve un ritorno dallo use case.

Elementi grafici del modello dei casi d'uso



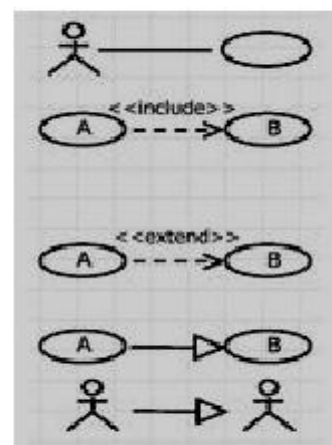
Attore: è qualcuno (utente) o qualcosa (sistemi esterni – dispositivi hardware) che:

- controlla le funzionalità;
- fornisce input o riceve output dal sistema;
- un attore modella un'entità esterna che comunica con il sistema;
- ogni attore ha un nome univoco.



Use Case: è un'unità funzionale parte del sistema. Un caso d'uso rappresenta una funzionalità offerta dal sistema, in termini di un flusso di eventi.Ogni caso d'uso ha un nome univoco.

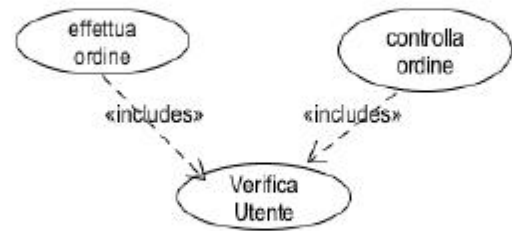
Relazioni principali



- **Association**: identifica relazioni semplici tra attori e casi d'uso.
- **Include**: fattorizza proprietà comuni specificando azioni in esso contenute come una sorta di sottoprocedure.
- **Extend**: identifica comportamenti simili (varianti). A può essere visto come una variante di B o come caso particolare.
- **Generalization**: si applica sia ad attori che a use case. A eredita il comportamento di B. A può essere sostituito ad ogni occorrenza di B.

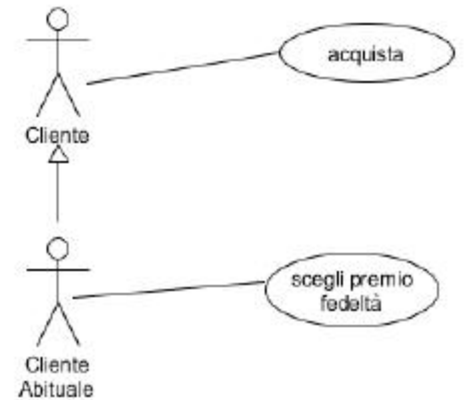
Rlalazione di Inclusione

Un comportamento comune a più casi d'uso diventa un caso d'uso che è incluso nei casi d'uso di partenza.L'inclusione avviene in un punto preciso della descrizione del caso d'uso includente. Il caso incluso non ha senso da solo.



Generalizzazione

Si indica con una linea continua e freccia con la punta non riempita. **Deve sempre valere il principio di sostituzione.**



Si può applicare: **Tra Attori**

La relazione di generalizzazione fra attori si applica quando un attore è un sottotipo di un altro e questo comporta delle differenze nel rapporto con il sistema. La relazione viene indicata da una freccia con la punta non riempita.

Tra Casi d'Uso

Un caso d'uso figlio eredita il compo-rtamento ed il significato del caso d'uso padre.
La relazione di generalizzazione fra use cases si ha quando un caso d'uso è un caso particolare di un altro, per questo è utile per rappresentare i percorsi alternativi di un'interazione complessa con il sistema.

Come detto, ogni use case è una lista di scenari, ed ogni scenario è una sequenza di passi. Per ciascun use case, ogni scenario avrà la sua propria pagina rappresentata nel seguente modo:

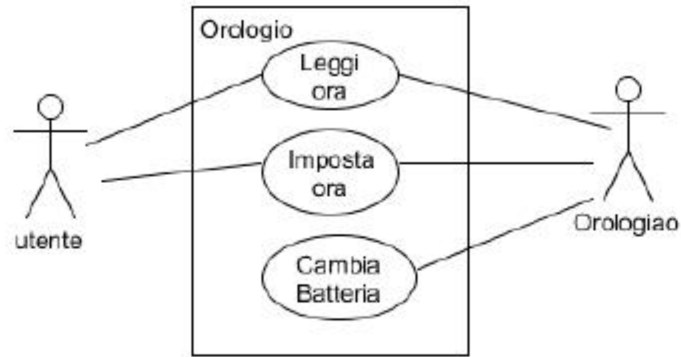
- **Un actor che dà inizio alla sequenza dell'use case**
- **Le Pre condizioni per lo use case**
- **I passi dello scenario vero e proprio**
- **Le Post Condizioni quando lo scenario è completo**
- **L'actor che beneficia dell'use case**

Gli Use Case Diagram danno del valore aggiunto alla raccolta di informazioni. Essi visualizzano le risposte che il programma deve fornire alle azioni operate dall'utente o agli eventi accaduti cioè il comportamento che il programma deve adottare rispetto alle possibili situazioni. Questi comportamenti po saranno i metodi degli oggetti che andremo ad utilizzare.

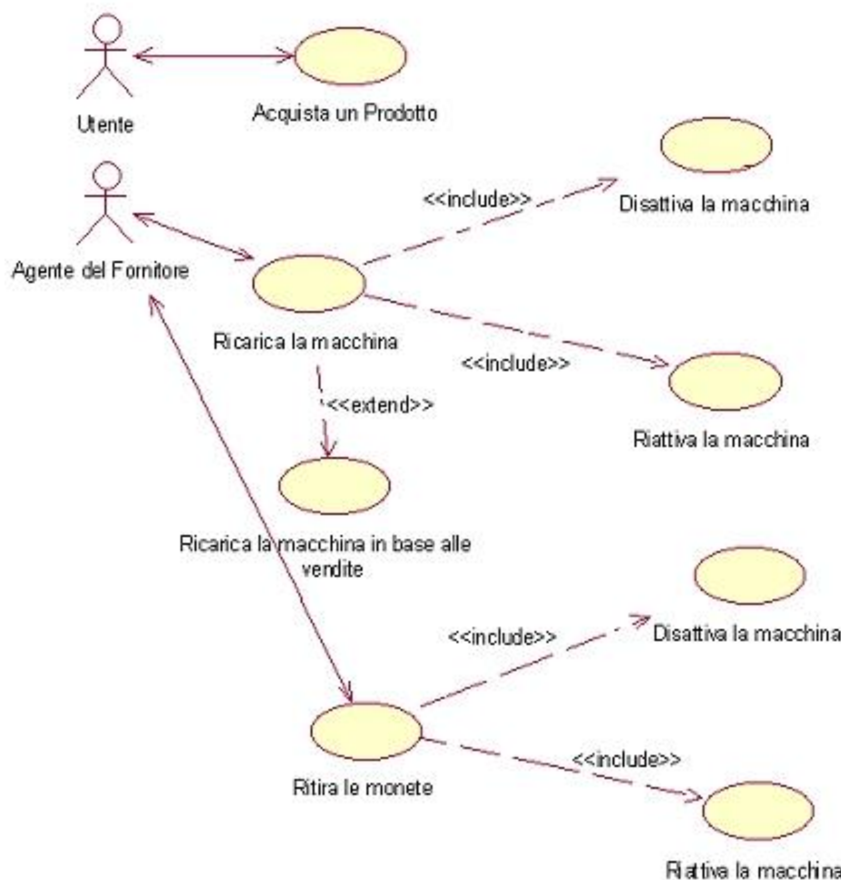
- ogni caso d'uso corrisponde a uno o più requisiti funzionali;
- ogni caso d'uso è coinvolto in una qualche relazione;

Essi Mostrano:

- le modalità di utilizzo del sistema (casi d'uso);
- gli utilizzatori e coloro che interagiscono con il sistema (attori);
- le relazioni tra attori e casi d'uso.



L'esempio della figura che segue mostra una possibile rappresentazione di un sistema automatico di distrubuzione di bevande:



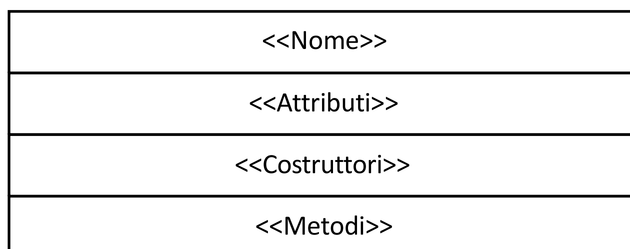
ovviamente potrebbe essere esteso il caso d'uso acquista prodotto perché per acquistare un prodotto l'utente deve inserire il denaro scegliere un prodotto prelevare il resto ecc.

Nella fase di disegno OOD(Object Oriented Design) , che segue l'individuazione dei casi d'uso, si devono individuare le classi (o le gerarchie di classi) necessarie, rispondendo alla domanda "chi fa che cosa?". Questa domanda può essere tradotta come "quali Classi devono fornire i servizi necessari per realizzare le funzionalità individuate nei casi d'uso?"

- Si decidono le caratteristiche informative, il tipo di dati e le caratteristiche relazionali che definiscono gli attributi (campi o data member) della classe.
- Si individuano quali servizi quelle classi devono fornire. Ovvero quali

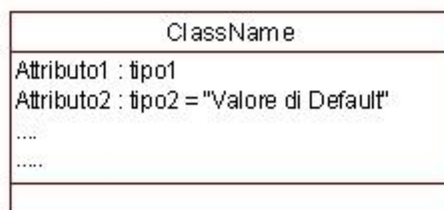
operazioni (metodi) sono necessarie per “manipolare” un oggetto di quella classe.
- Si definiscono i costruttori della classe.
- Si individuano le relazioni tra le classi progettate.

Sotto è disegnato lo schema di rappresentazione di un Classe.



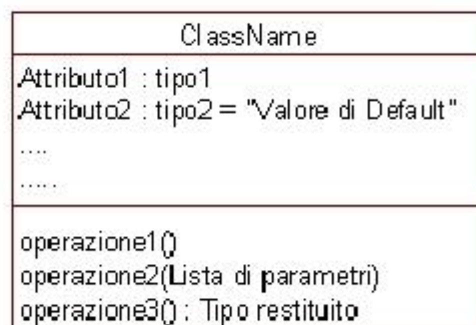
Definizione generica di una classe

Un attributo rappresenta una proprietà di una classe. Esso descrive un insieme di valori che la proprietà può avere quando vengono istanziati oggetti di quella determinata classe. Una classe può avere zero o più attributi.
Un **Attributo** il cui nome è costituito da una sola parola viene scritto sempre in caratteri minuscoli. Se, invece, il nome dell’attributo consiste di più parole (es: Informazioni-Cliente) allora il nome dell’attributo verrà scritto unendo tutte le parole che ne costituiscono il nome stesso con la particolarità che la prima parola verrà scritta in minuscolo mentre le successive avranno la loro prima lettera in maiuscolo. Nell’esempio appena visto l’attributo sarà identificato dal termine: informazioniCliente.
La lista degli attributi di una classe viene separata graficamente dal nome della classe a cui appartiene tramite una linea orizzontale.



Nell’icona della classe, come si vede nella figura precedente, è possibile specificare un tipo in relazione ad ogni attributo (string, float, int, bool, ecc.). E’ anche possibile specificare un valore di default che un attributo può avere.

Un’**Operazione** è un’azione che gli oggetti di una certa classe possono compiere. Analogamente al nome degli attributi, il nome di un’operazione viene scritto con caratteri minuscoli. Anche qui, se il nome dell’operazione consiste di più parole, allora tali parole vengono unite tra di loro ed ognuna di esse, eccetto la prima, viene scritta con il primo carattere maiuscolo. La lista delle operazioni (metodi) viene rappresentata graficamente sotto la lista degli attributi e separata da questa tramite una linea orizzontale.



Anche I metodi possono avere delle informazioni addizionali. Nelle parentesi che seguono il nome di un’operazione, infatti, è possibile mostrare gli eventuali parametri necessari al metodo insieme al loro tipo. Infine, se il metodo rappresenta una

funzione è necessario anche specificare il tipo restituito.

Come si può riuscire a individuare le classi da utilizzare dalla intervista con il cliente? O dal testo del problema assegnato?

E' necessario, a tal fine, prestare particolare attenzione ad i **nomi** che i clienti usano per descrivere le entità del loro business. Tali nomi saranno ottimi candidati per diventare delle **classi** nel modello UML. Si deve prestare, altresì, attenzione ai **verbi** che vengono pronunciati dai clienti. Questi costituiranno, con molta probabilità, i metodi (le operazioni) nelle classi definite.

Una classe che non si interfacci con altre classi è sicuramente poco significativa in OOP. Abbiamo visto che gli oggetti, in un programma Object Oriented, interagiscono tra loro utilizzando lo scambio di messaggi per richiedere l'esecuzione di un particolare metodo.
Tale comunicazione consente di identificare all'interno del programma una serie di relazioni tra le classi in gioco la cui documentazione risulta essere assai utile in fase di disegno e di analisi.

Le più comuni relazioni tra classi, in un programma ad Oggetti sono identificabili in tre tipologie:

- **Associazioni** (Use Relationship)
- **Aggregazioni** (Containment Relationship)
- **Specializzazioni** (Inheritance Relationship)

Associazione simbolo: freccia orientata

L'Associazione è il tipo di Relazione più intuitiva ed anche più diffuso. In generale, diciamo che una classe A utilizza una classe B se un oggetto della classe A è in grado di inviare dei messaggi ad un oggetto di classe B oppure se un oggetto di classe A può creare, ricevere o restituire oggetti di classe B.
Più in dettaglio, potremmo dire che una classe è associata ad un'altra se è possibile "navigare" da oggetti della prima classe ad oggetti della seconda classe seguendo semplicemente un riferimento ad un oggetto.(facendo una istanza dell'oggetto)

Ad **esempio**, dato un oggetto di tipo Persona, è possibile giungere ad oggetti di tipo Azienda accedendo semplicemente alla variabile istanza azienda definita all'interno della classe Persona.. Nella figura seguente viene rappresentata graficamente la relazione di tipo Associazione tra queste due classi:

Figura 1. Diagramma di una relazione di associazione

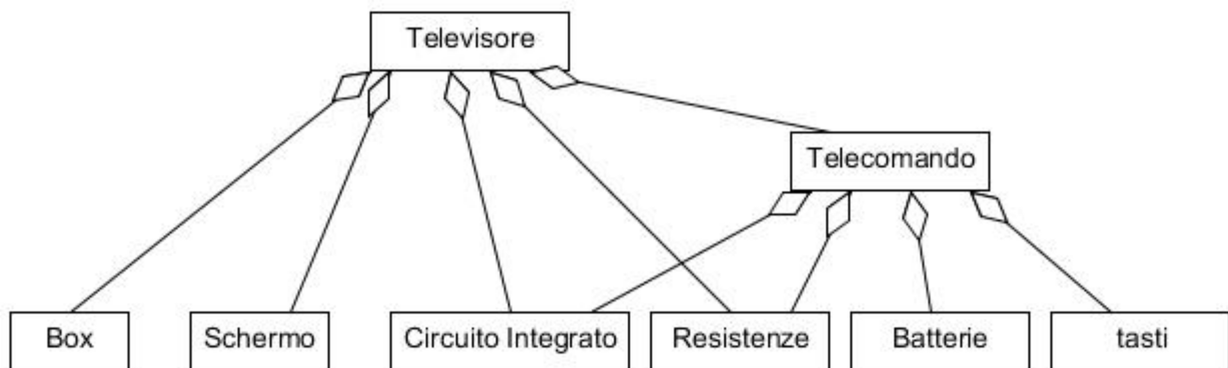
Si può notare, osservando la figura, come sia anche possibile assegnare un nome alla associazione tra le due classi (Il nome in questione è: "Lavora per") e dare una direzione all'associazione stessa, intendendo con ciò il verso in cui avviene la

navigazione tra le classi.
Se la navigazione è, invece, possibile in entrambe le direzioni si parlerà di **associazione bidirezionale** e non si inserirà alcuna freccia..

Aggregazione Simbolo:

La relazione di tipo Aggregazione si basa, invece, sul seguente concetto: Un oggetto di classe A contiene un oggetto di classe B se B è una proprietà (attributo) di A.
In sostanza, l'aggregazione è una forma di associazione più forte: una classe ne aggrega un'altra se esiste tra le due classi una relazione di tipo "intero-parte".
Ad esempio la classe Azienda aggrega la classe Persona perché una ditta (che costituisce l'"intero") è composta da persone (che costituiscono la "parte").
Una classe ContoBancario, invece, non è legata da una relazione di tipo aggregazione con la classe Persona anche se può essere plausibile che sia possibile navigare da un oggetto che rappresenta un conto bancario fino ad un oggetto che rappresenta una persona, il proprietario del conto. Dal punto di vista concettuale, però, una persona non si può far appartenere ad un conto bancario.

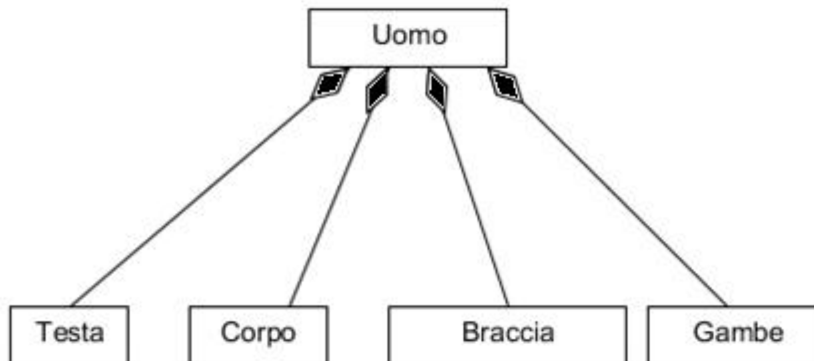
Un esempio di aggregazione
Esaminiamo le "parti" che costituiscono un Televisore. Ogni TV ha un involucro esterno (Box), uno schermo, degli altoparlanti, delle resistenze, dei transistors, un circuito integrato e un telecomando (oltre, naturalmente, ad altri tantissimi componenti!). Il telecomando può, a sua volta, contenere le seguenti parti: resistenze, transitors, batterie, tastiera ecc.
Il Class Diagram che ne deriva sarà il seguente:



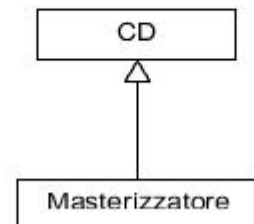
Composizione Simbolo:

La Composizione è una forma di aggregazione ancora più forte che indica che una "parte" può appartenere ad un solo "intero" in un certo istante di tempo.
Ad esempio, un pneumatico può far parte di una sola automobile in un certo istante, mentre, al contrario, una persona potrebbe lavorare contemporaneamente per due ditte.
Ad essere sinceri, le differenze tra associazione, aggregazione e composizione possono trarre in confusione anche gli analisti più esperti. Per tale motivo si raccomanda di utilizzare tali tipi di relazioni soltanto se il loro utilizzo può essere di reale giovamento.

Il simbolo utilizzato per una composizione è lo stesso utilizzato per un’aggregazione eccetto il fatto che il rombo è colorato di nero.
Se, ad esempio, si esamina l’aspetto esterno degli uomini, si evincerà che ogni persona ha (tra le altre cose): una testa, un corpo, due braccia. e due gambe. Tale concetto viene rappresentato nel seguente grafico:



Specializzazione Simbolo:
La relazione di tipo Specializzazione si basa sul concetto di **ereditarietà.** Un oggetto di classe A deriva da un oggetto di classe B se A è in grado di compiere tutte le azioni che l’oggetto B è in grado di compiere.
Inoltre l’oggetto di classe A è in grado di eseguire anche azioni che l’oggetto B non può compiere. Ad esempio, un masterizzatore rappresenta anch’esso un tipo di lettore CD (poiché riesce anche a leggere CD), e volendo si potrebbe pensarlo come una estensione di quest’ultimo. In più, rispetto a quest’ultimo, ne estende le funzionalità, visto che è in grado anche di scrivere sui supporti ottici.



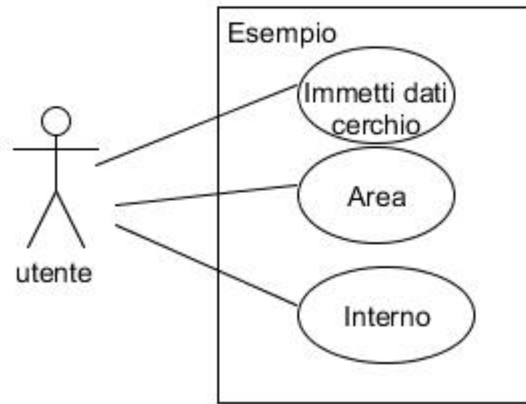
Esempi di progettazione

Supponiamo ad esempio di voler affrontare la semplice situazione problematica seguente:
esempio 1 - "Si desidera realizzare un sistema che manipoli cerchi calcolandone l'area e determinando se un punto assegnato è interno o esterno al cerchio."

Nella fase di analisi (OOA) si individueranno le esigenze dell'utente che si possono desumere dalle parole chiave del testo:

- possibilità di inserire i dati che definiscono un cerchio;
- possibilità di calcolare l'area:
- possibilità di sapere se un punto assegnato è interno al cerchio.

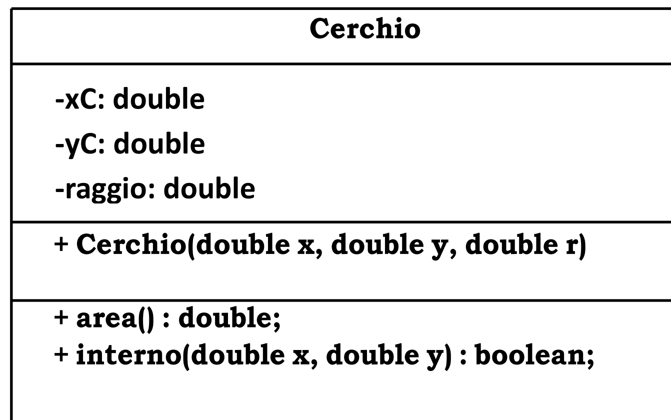
Questi sono i tre casi d'uso che possiamo stabilire in prima approssimazione e schematizzare con il seguente diagramma:



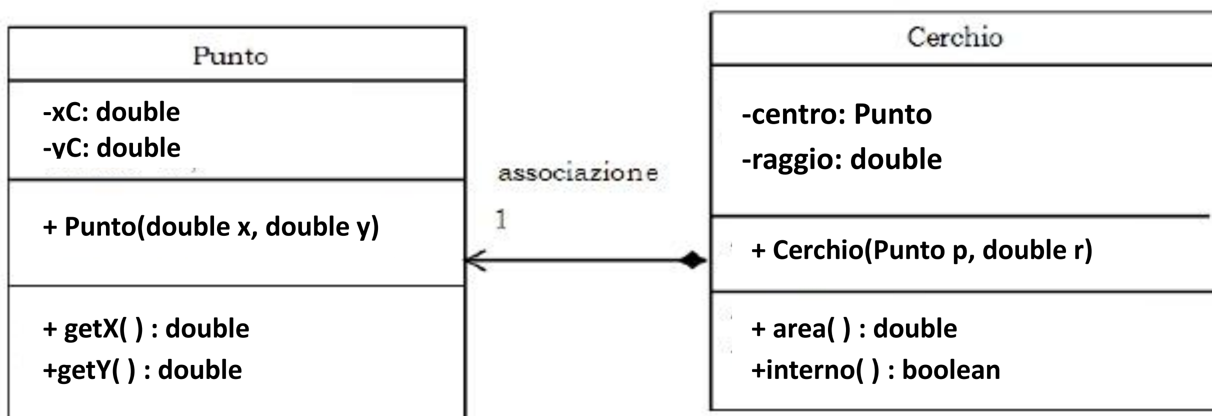
Nella fase di **Disegno (OOD)** si dovranno definire le classi necessarie CHE FANNO ciò che si desidera (CHI FA CHE COSA?).
Per il nostro problema la classe centrale sarà il cerchio e dovrà avere come attributi ***raggio* e *centro***. I metodi minimi necessari saranno:

- **il costruttore che consente l'immissione dei dati;**
- **il metodo Area;**
- **il metodo interno.**

Lo schema da realizzare per disegnare tale classe è il seguente:

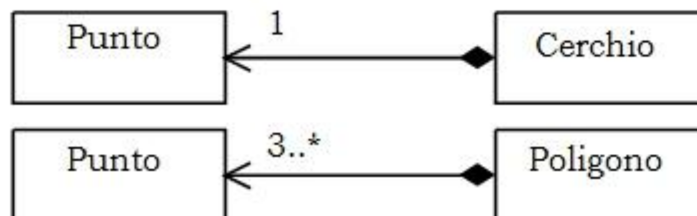


Si potrebbero progettate non una ma due classi: la Cerchio, indispensabile, e una classe Punto che potrebbe indicare il fatto che il cerchio è dotato di un punto particolare che è il suo centro. In questo modo il disegno alternativo sarebbe:



Le due alternative di disegno sono didatticamente utili per mostrare come si rappresenta una relazione tra due diverse classi. E' evidente che se si progetta una classe Punto si deve prevedere un costruttore adeguato e che per accedere dall'esterno agli attributi privati del Punto dovranno essere disegnati i due metodi getX() e getY().

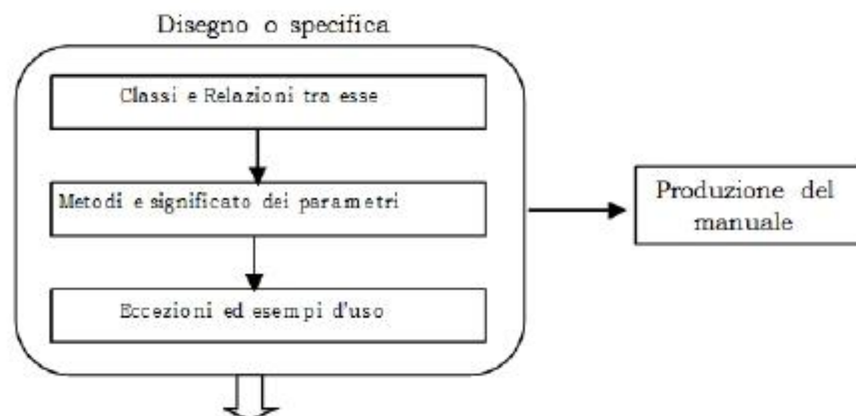
Annotazioni teoriche:
Modalità con cui si indicano attributi, costruttori e metodi. Il segno che li precede (-), (+), (#) indica rispettivamente che il metodo è *private*, *public* o *protected*. Un metodo o un attributo public (+) è SEMPRE accessibile dall'esterno della classe, un metodo o un attributo private (-) è accessibile SOLO dal codice interno alla classe; infine un metodo o un attributo protected (#) è accessibile solo dal codice della classe o delle sue sottoclassi ma non dall'esterno.
Le classi possono stare tra loro in una relazione associativa di Composizione e questo si indica con la seguente simbologia



due classi sono in relazione di composizione se la prima di esse risponde alladomanda "HA UN" e la seconda di conseguenza si configura come la parte di un tutto. Nel primo caso disegnato si dice che la classe Cerchio ha un Punto (ilcentro) che le appartiene come sua parte. Nel secondo caso disegnato la relazione ci informa del fatto che un Poligono deve avere come sue parti 3 o più punti.

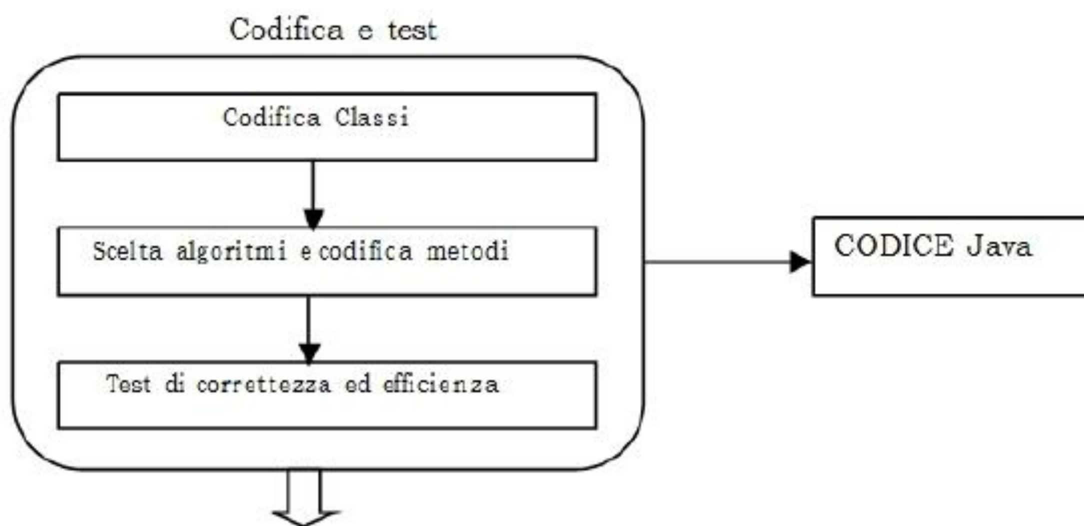
Sempre nella fase didisegno (ma potrebbe essere realizzata contemporaneamente alla codifica In Eclipse vi è un ottimo tool per produrre la documentazione basta scrivere opportunamente i commenti) si deve definire la Documentazione della classe; questa consiste nello scrivere le specifiche o manuale d'uso di ogni metodo della classe.
Per queste finalità tale manuale dovrà contenere una "intestazione" dei metodi (detta anche *signature*) scritta nella sintassi del linguaggio di programmazione e la descrizione per ogni metodo di:

- effetti prodotti dal metodo;
- descrizione dei parametri di input e di output;
- casi d'uso ed eventuali eccezioni.



Nome	Identificatore	Cerchio
Campi o attributi	Punto centro double raggio	private
Costruttori	*Intestazione:*	Cerchio(Punto p, double r)
	Invocazione:	Punto p=new Punto(3,4); Cerchio C=new Cerchio(p, 9);
	Effetti:	Riceve il centro p di coordinate (3,4) e il raggio 9 e costruisce un oggetto cerchio C.
Metodi	*Intestazione:*	double area()
	Invocazione:	double S=C.area();
	Effetti::	determina l'area del cerchio C e l'assegna alla variabile S.
	Intestazione:	boolean interno(Punto p)
	Invocazione:	boolean b=C.interno(p);
	Effetti::	determina se il punto p assegnato è o no interno al cerchio C. Restituisce true se Si, No altrimenti.

Nella fase di codifica (OOP) delle Classi e dei metodi (attività specifica del programmatore di sistema) si deve scegliere l'algoritmo più opportuno per implementare ciascun metodo. Di norma questa fase è lasciata libera da vincoli sia nella scelta delle strutture dati di supporto più opportune che nella sceltadell'algoritmo; ovviamente tale scelta deve superare i test di funzionamento previsti dalle specifiche e gli eventuali test di efficienza (la velocità di esecuzione dipende dalla rappresentazione dei dati e dall'algoritmo scelti).



Nel caso del problema da cui siamo partiti si tratterà di codificare le due Classi individuate nel disegno e di costruire un programma di prova che verifichi il funzionamento dei Casi d'Uso previsti.

```
public class Punto {
      private double xC;
      private double yC;

   public Punto(double x, double y) {
      xC=x; yC=y;
      }
   public double getX() {
      return xC;
      }
   public double getY() {
      return yC;
      }
   } // end Punto

public class Cerchio {
      private Punto centro;
      private double raggio;
   public Cerchio(Punto p, double r) {
      centro=p; raggio=r;
      }

   public double area() {
      double ris=Math.PI*raggio*raggio;
      return ris;
      }

   public boolean interno(Punto p) {
      boolean ris=false;
      double x=p.getX();
      double y=p.getY();
      double xC=centro.getX();
      double yC=centro.getY();
      double dist=Math.sqrt((x-xC)*(x-xC)+(y-yC)*(y-yC));
      if (dist<=raggio) ris=true;
      return ris;
      }
   } // end Cerchio
```

Il main() deve verificare i "casi d'uso" individuati nell'analisi, in particolare calcolare l'area e stabilire se un punto assegnato è o no interno al cerchio:

```
public class Programmma
{
   public static void main(String arg[])
  {
      Punto Centro=new Punto(3,4);
      Cerchio C=new Cerchio(Centro, 9);
      Punto P=new Punto(3,13);
      Punto Q=new Punto(4,10);
      Punto R=new Punto(5,13);
      System.out.println("Area="+C.area());
      System.out.println("P interno "+C.interno(P)+", Q interno "    +
         C.interno(Q) +", R interno "+C.interno(R));
   }
}
```

Il programma stamperà:

Area=254.46900494077323
P interno true, Q interno true, R interno false