Progettazione del software

- Riguarda tutte quelle attività che permettono di passare dalla raccolta ed elaborazione dei requisiti di un sistema software alla sua effettiva realizzazione

- Ponte tra la fase di ***specifica*** e la fase di ***codifica***

- Durante la fase di progettazione si decidono le modalità di passaggio da "che cosa" deve essere realizzato (specifica) a "come" la realizzazione deve avere luogo

- La complessità della progettazione viene "ridotta" suddividendo il sistema complessivo in più sottosistemi

- ***Vantaggi***:
  - ***complessità*** delle singole parti ***minore*** della complessità totale originaria;
  - i sottosistemi ottenuti possono essere ***realizzati*** ed analizzati da gruppi diversi di programmatori in modo il più possibile ***indipendente***

- Due esigenze contrastanti:
  - ***progetto*** risultante ***sufficientemente astratto*** per poter essere agevolmente confrontato con le specifiche da cui viene derivato
  - ***progetto sufficientemente dettagliato*** in modo tale che la codifica possa avvenire senza ulteriori necessità di chiarire le operazioni che devono essere realizzate

- A seconda della tecnica impiegata per la progettazione, la realizzazione del sistema può risultare più o meno naturale ed immediata

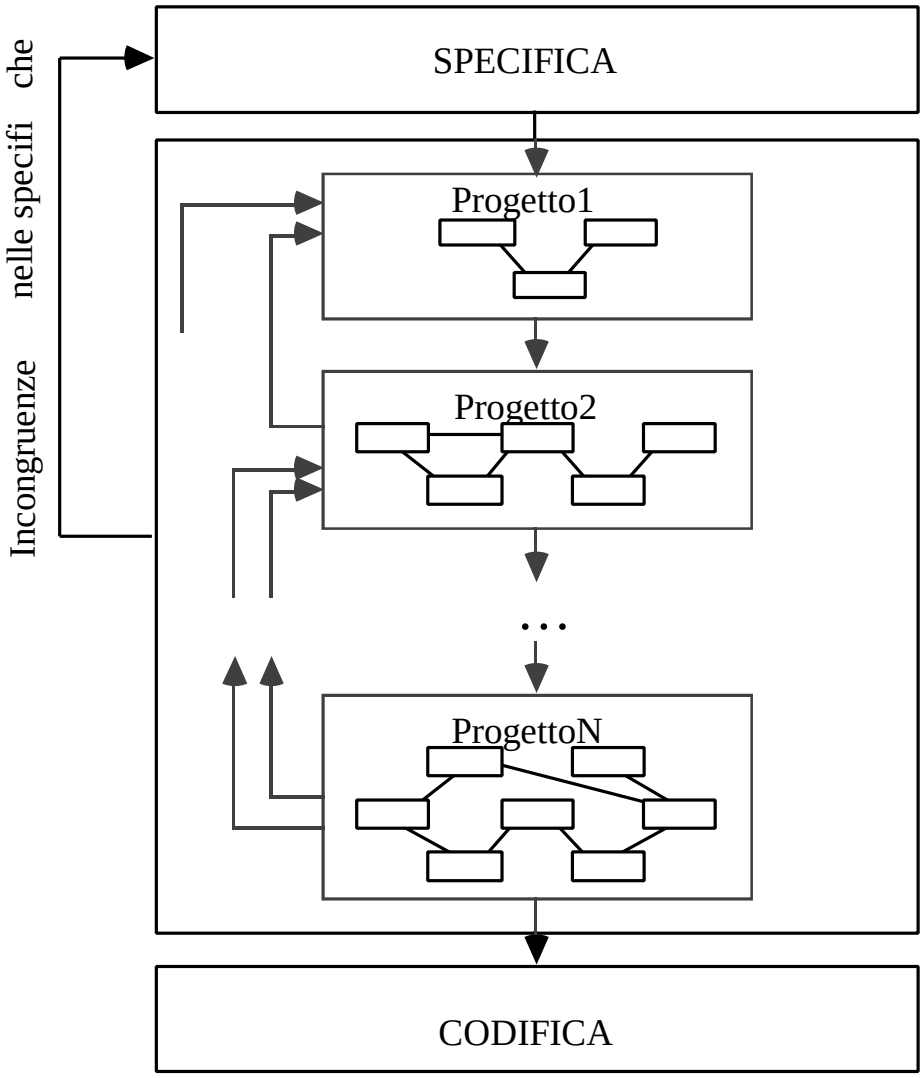
  Ad es., progettazione orientata a oggetti

  ☞ *realizzazione in un linguaggio a oggetti*

- Il progetto di un sistema software non si ottiene in tempi brevi ne` in un solo passo

- ***Sottosistemi*** individuati troppo complessi per essere realizzati direttamente: ***iterazione*** della ***suddivisione*** fin dove necessario

Iterazione del metodo



- Non esiste un criterio che permetta di individuare con certezza fino a dove sia necessario spingere il livello di dettaglio
- Regole empiriche basate sulla dimensione dei sottosistemi

- Non esiste un metodo generale per la progettazione del software

- Tipologie di software (software sequenziale, concorrente ed in tempo reale)

- In fase di ***progettazione*** vengono fissate le ***direttive di sviluppo*** del software, le quali costituiscono un riferimento che il più delle volte risulta particolarmente vincolante per le attività successive (***scelte di progetto***)

- Ad una ***stessa specifica*** possono corrispondere ***più progetti***, ossia più metodi di soluzione diversi

- Le scelte di progetto devono poter cambiare in risposta a mutate esigenze di vario tipo senza che per questo tutto il progetto e perciò tutto il software prodotto debba essere modificato radicalmente

- Scelte di progetto espresse e realizzate in modo che il progetto ne risulti *trasparente*, ossia suffi-cientemente indipendente

- Il progetto di un sistema software è perciò un'attività altamente *creativa*, che richiede un insieme di abilità a coloro che vi sono preposti

Requisiti per il progettista

- Profonda conoscenza di tutto ciò che riguarda il processo di sviluppo del software

- Capacità di saper *anticipare i cambiamenti* (modifiche effettive in porzioni limitate e ben identificate del sistema software prodotto, senza che ne venga intaccata la struttura complessiva)

- Notevole *inventiva* per riuscire a trovare una soluzione progettuale accettabile anche in mancanza di una metodologia che sia sufficientemente espressiva

- Buon grado di *esperienza* per poter individuare con maggiore rapidità e sicurezza le soluzioni più opportune (allocazione di risorse)

Obiettivi della progettazione

- Produrre software con le ***caratteristiche di qualità*** che sono state dettagliate nella fase di analisi e specifica dei requisiti

- Ad esempio:
  - affidabilità
  - modificabilità
  - comprensibilità
  - riusabilità

- Obiettivi che si possono riassumere nella ***diminuzione dei costi e tempi*** di produzione e nell'***aumento della qualità*** del software

- I costi maggiori riguardano la fase di manutenzione del software

- Capacità di poter far fronte a modifiche da effettuare senza che l'intera struttura dell'applicazione già costruita debba essere messa nuovamente in discussione ed elaborata

- Saper anticipare il cambiamento durante la progettazione: ciò si ha, secondo la dizione di Parnas,

con la cosiddetta *progettazione per il cambiamento* (*design for change*)

Affidabilita`

- Durante la progettazione viene effettuata una verifica approfondita del fatto che le specifiche siano effettivamente dotate di tutte le necessarie caratteristiche di consistenza, completezza, etc.

- Con la suddivisione in sottosistemi del sistema complessivo, verifica dei sottosistemi

- Collegamenti tra sottosistemi semplici e regolari, per cui anche i difetti nelle interrelazioni tra i sottosistemi risultano meno frequenti e di minore entità

Modificabilita`

- Cambiamenti nel sistema di calcolo per cui il prodotto viene sviluppato

Macchine fisiche e sistemi operativi differenti, oppure anche sistemi per la gestione di basi di dati diversi, etc

- Evoluzione nel tempo delle specifiche dei requisiti

L'uso dell'applicazione mostra la possibilità o la necessità di modificare le procedure impiegate

- Cambiamenti nel software per il miglioramento delle prestazioni

Modifica del linguaggio di programmazione scelto (inizialmente, costruzione del prototipo, anche inefficiente, ad esempio Smalltalk; poi realizzazione in C++)

Modifica degli algoritmi

Modifica delle strutture dati utilizzate

- Inadeguatezza della specifica dei requisiti

- Evoluzione per motivi di mercato

Comprensibilità del progetto

- Un progetto comprensibile, con un'adeguata documentazione, può essere rivisto ed utilizzato anche da persone che non hanno partecipato direttamente alla sua stesura

- Aumenta le caratteristiche di affidabilità e di modificabilità

Riusabilità

- Poter utilizzare un'applicazione od anche una sua parte all'interno di una nuova applicazione

- Sottosistemi costruiti in modo tale da poter essere riutilizzati

- Diminuzione dei costi e dei tempi di produzione

- La parte di software che viene riutilizzata dopo essere stata sottoposta a convalida e ad un periodo di uso operativo è più affidabile

- Costruire o comperare componenti software (*make or buy*)?

- Il riuso molto spesso si accompagna ad una modifica del software

Criteri generali e metodologie: **moduli ed architettura**

- Il termine *modulo* designa ciascuno dei sottosistemi in cui il sistema complessivo viene suddiviso
- La progettazione di un sistema software riguarda l'identificazione dei vari moduli e delle loro mutue interrelazioni
- Modulo: sottoprogramma, ma anche un'unità a sé stante
- Un modulo risulta quindi uno strumento per raggruppare tipi, strutture dati, sottoprogrammi etc
- Contiene e fornisce risorse e servizi computazionali
- La definizione di un modulo deve essere basata su due fattori fondamentali:
  - il grado di *coesione* del suo contenuto
  - il grado di *disaccoppiamento* con il resto del sistema

Moduli

- **Coesione interna**

Unità sostanzialmente ***omogenea*** e facilmente comprensibile

Ciò che è contenuto in un modulo deve far riferimento ad un insieme di ***caratteristiche comuni***

Maggior comprensibilità

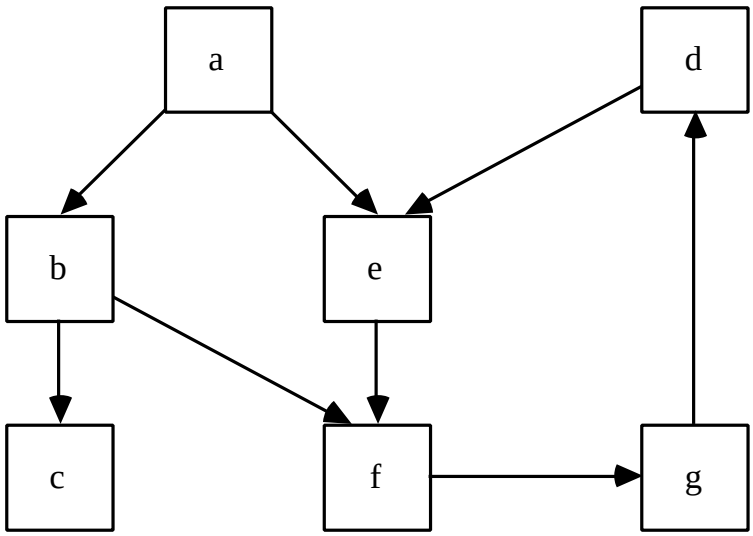
Maggiore facilità nella realizzazione e nella modifica

- ***Grado di disaccoppiamento***

Maggiore è il disaccoppiamento, tanto più si è riusciti a dominare la ***complessità*** del sistema totale

Architettura del sistema

- Struttura del sistema complessivo
- Indica quali siano le ***interrelazioni*** tra i vari moduli
- Esistono più tipi di relazioni che possono intervenire tra i moduli
- Relazioni tra moduli rappresentate in forma matematica o mediante grafi orientati



Relazione USA

- m_i USA m_j (definita staticamente)
- Affinché il modulo m_i risulti corretto rispetto alle sue specifiche, è necessaria anche la corretta esecuzione di m_j
- Modulo m_i cliente di servizi forniti dal modulo m_j (di solito attraverso chiamata di procedure)
- Sia $M = \{m_i \mid i=1,..,n\}$

  USA = $M \infty M$ disaccoppiamento troppo basso

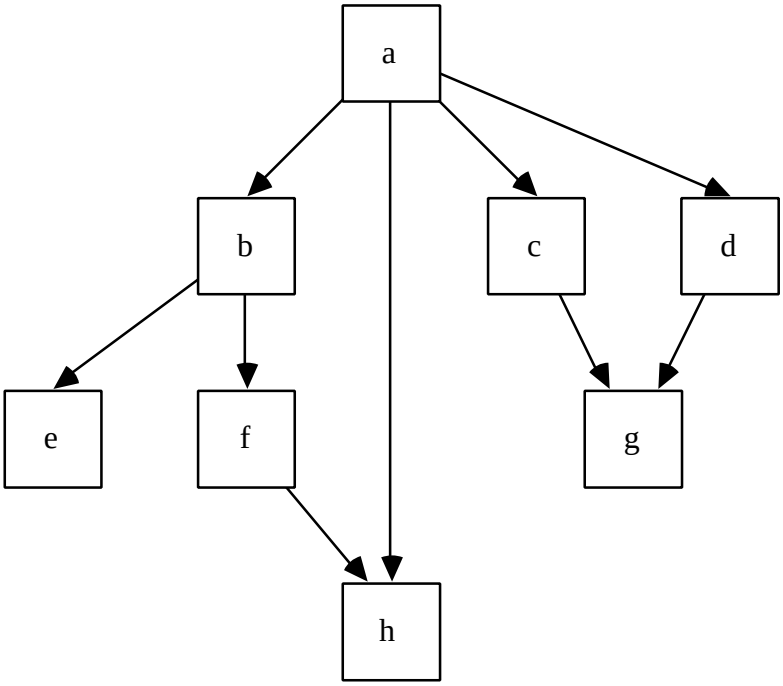
  USA = Ø funzionalità comuni a più moduli, distribuite e replicate (scarsa coesione)

- Di solito si richiede che sia aciclica (gerarchia)

Relazione COMPONENTE_DI

- Raffinamento progressivo dei moduli in sottomoduli (approccio *top-down*)

- Deve necessariamente essere costruita in modo tale da dar luogo ad un grafo orientato aciclico (***gerarchia***)



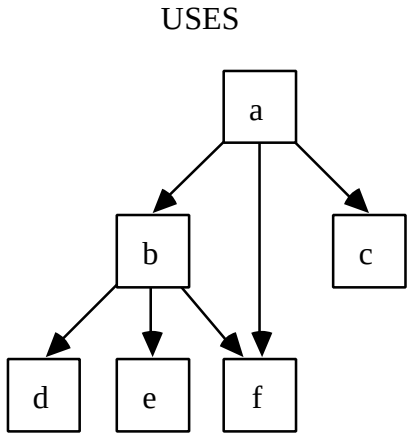
- Solo i moduli con grado di uscita 0 avranno un'esistenza fisica effettiva nel sistema software finale

- La scomposizione così effettuata può servire a fini di documentazione del progetto

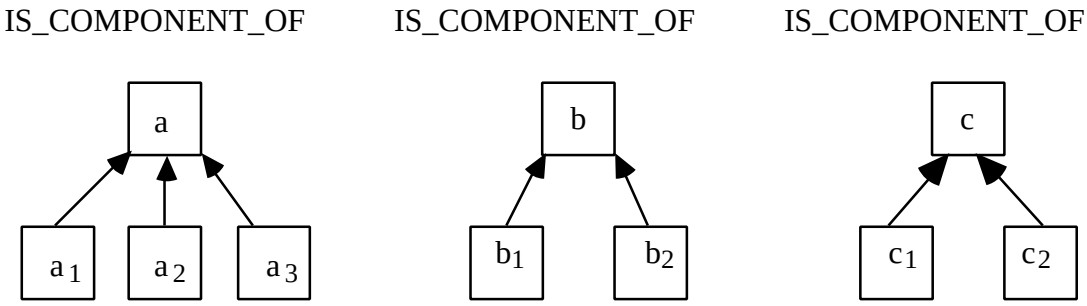
- Modulo m_i scomposto in una famiglia di sottomoduli m(i)

- Se m_i USA m_j, ridefinizione di vari collegamenti dai moduli di m(i) a m_j

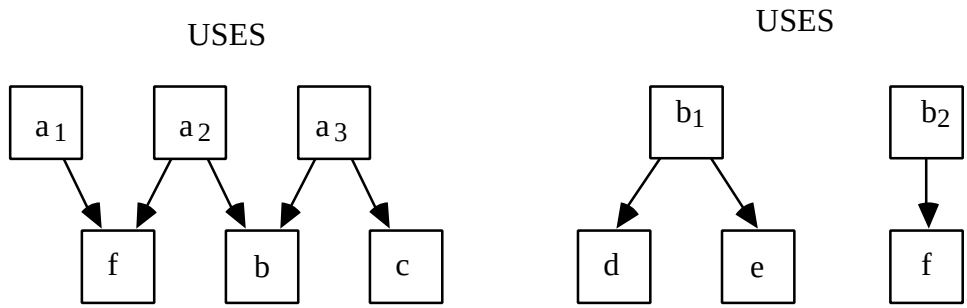
Esempio



Ciascuno dei moduli *a*, *b* e*c* viene raffinato in moduli componenti



Legami dovuti alla relazione USA sui moduli componenti:



Non è fornita la ridefinizione di USA dal momento che il modulo *c* non era in tale relazione con alcun altro modulo, per cui non lo sono neanche c_1 e c_2

Costruzione incrementale

- Metodologia particolarmente efficace quando si intenda sviluppare un prodotto software a partire da un prototipo

- Il primo prototipo realizza le funzionalità più importanti per dimostrare la fattibilità

- Aggiunta di funzionalità, oppure riscrittura delle funzionalità già realizzate al fine di migliorarne efficienza e precisione

Differimento delle decisioni

- Le scelte di progetto devono essere effettuate solo quando si raggiunga una conoscenza necessaria e sufficiente per operarle

- Attivita` condotta con l'ausilio del principio di information hiding

- Si limita l'entità degli inconvenienti derivanti da una decisione non corretta al singolo modulo

Information hiding

- Occorre stabilire quale sia il "contenuto" di ogni modulo e quali siano i "servizi" che esporta

- Ogni modulo deve presentare al suo esterno un insieme di funzionalità che sia il più possibile stabile

- Il contenuto del modulo deve invece poter essere deciso localmente al modulo

- Obiettivo finale della progettazione: definizione di un ***insieme di moduli*** che devono essere visti dall'esterno come scatole nere, accessibili solo tramite i ***servizi*** che mettono a disposizione

- Un modulo mette a disposizione degli altri moduli una ***interfaccia***

- Il contenuto del modulo prende il nome di ***realizzazione***

Cosa nascondere: scelte di progetto

- ***Algoritmo***

  Visibile all'esterno soltanto attraverso il nome della funzione (*sottoprogramma*, *metodo*) che lo realizza

  Il modulo *cliente* non deve conoscere i dettagli del procedimento scelto dal modulo servitore per la realizzazione di una funzionalità (maggiore disaccoppiamento)

- ***Struttura dati***

  La scelta della struttura dati deve essere anch'essa tenuta nascosta ai fini di migliorare la modificabilità

  Ad esempio, tabella come vettore, lista, file, etc.

- ***Politiche***

  Accessi a risorse condivise

  La politica di gestione scelta deve essere del tutto invisibile all'esterno

Architettura modulare: vantaggi

- Rapida prototipazione del sistema

- Definita l'architettura del sistema e specificate le interfacce dei vari moduli, la realizzazione dei singoli componenti può essere fatta senza curare in modo particolare l'efficienza

- Il sistema può poi essere progressivamente modificato con l'obiettivo di ingenerizzare la realizzazione dei requisiti

Notazioni di progetto

- Anche la progettazione porta alla generazione di uno o più documenti che nel complesso costituiscono il progetto vero e proprio del software

- Lo scopo di tale documentazione è da un lato di fornire un riferimento univoco per le successive fasi di codifica e di verifica, e dall'altro di poter essere utilizzata per chiarire se durante la fase di progettazione non si siano avuti errori od omissioni nei riguardi di quanto stabilito durante la specifica

- Alla documentazione di progetto si fa riferimento nel caso si debbano apportare modifiche di una qualche entità al prodotto, purché queste non intacchino le specifiche

- Notazioni di progetto formali (***metodi algebrici***) o piu` pragmatici

- Notazioni che fanno riferimento all'uso di opportuni costrutti di linguaggi di programmazione (***testuali***) e all'uso di raffigurazioni visive (***grafiche***)

Esempio:
Textual Design Notation (TDN)

- Derivata dal linguaggio Ada
- Notazione (semi)formale per specificare un modulo

```
module M;
-- commento in generale sul modulo M: ad esempio gestisce un elenco
-- anagrafico
uses A, B, C;
exports     const       Max = 7;
        -- commento su Max: ad esempio è il numero massimo di persone
        type        Vet = array [1 .. Max] of Persona;
        -- commento su Vet: è il tipo dell'elenco gestito da M
        -- Persona è un tipo esportato dal modulo A
        var         X: integer;
        -- commento sul significato di X: è il numero di persone presenti
        -- inoltre può essere precisato che la variabile non è inizializzata
        -- in M
        procedure Y(in Par1: Vet; inout Par2: integer; out Par3: real);
        -- commento sul significato di Y: è da notare che il parametro
        -- Par1 è
        -- in ingresso, Par2 in ingresso uscita e Par2 solo in uscita
implementation
        -- eventuali commenti su strategie e motivazioni dell'implementazione
composed of     P, Q, R;
end M;
```

- L'***interfaccia*** del modulo può essere vista come divisa in sue sottosezioni: le ***importazioni*** e le ***esportazioni***

- Il modulo M, tramite l'utilizzo della clausola ***uses***, importa le risorse computazionali messe a disposizione dai moduli A, B e C

- Tramite l'utilizzo della clausola ***exports***, il modulo M esporta le risorse computazionali listate di seguito (tutto ciò che precede la parola chiave ***implementation***)

- Nella parte di ***implementazione*** vengono descritti i moduli componenti M, ossia P, Q e R

- Il modulo M contiene i moduli P, Q ed R:

```
module P;
-- …
uses A, R;
exports     const       Max = 7;
        -- …
        type        Vet = array [1 .. Max] of Persona;
        -- …
implementation
        -- …
end P;
```

```
module Q;
-- …
uses C, R;
exports     var         X: integer;
        -- …
        procedure Y(in Par1: Vet; inout Par2: integer; out Par3: real);
        -- …
implementation
        -- …
end Q;
```

```
module R;
-- …
uses A, B;
exports     …;
        -- …
        …;
        -- …
implementation
```

```
        -- …
end R;
```

Graphical Design Notation (GDN)

- Insieme all'uso di una notazione testuale il progetto di un prodotto software può essere sviluppato con l'ausilio di un ***formalismo grafico***

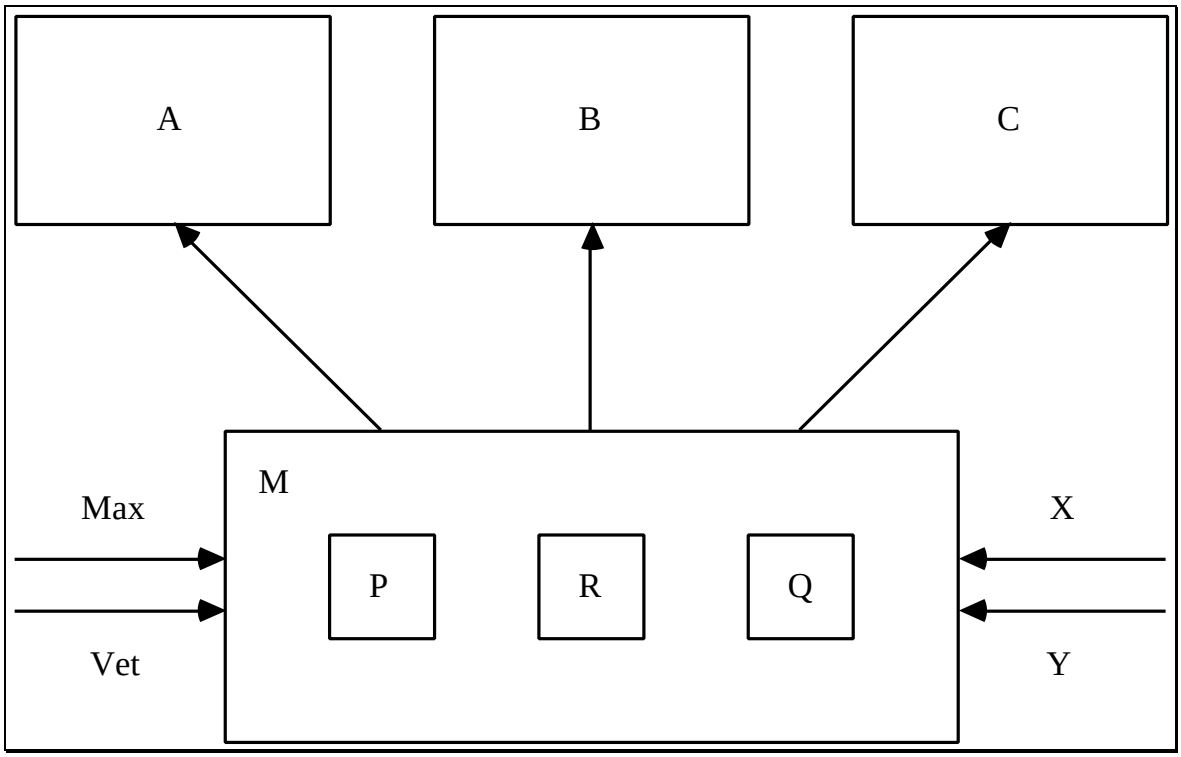
- Maggiore immediatezza nella rappresentazione grafica

- Notazione basata su un grafo aciclico che descrive le interrelazioni tra i moduli (relazioni USA e COMPONENTE_DI)

- Ogni ***nodo*** del grafo rappresenta un ***modulo***

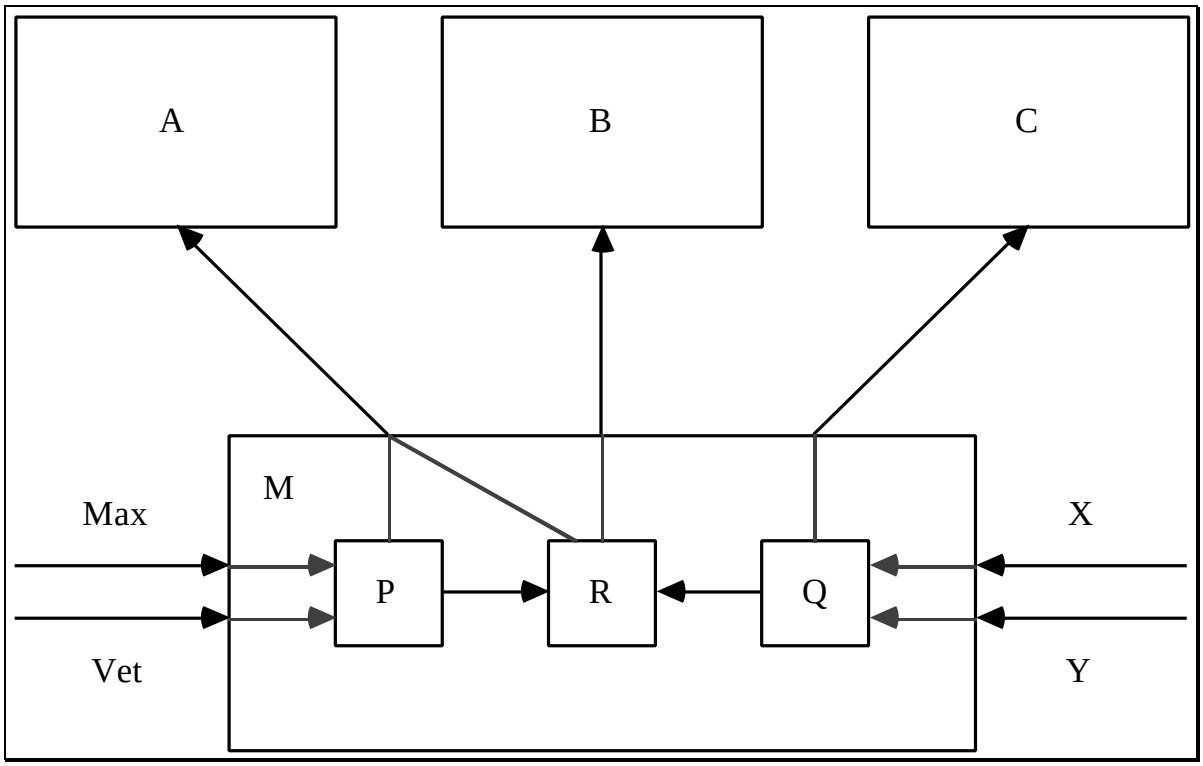
- Ogni nodo è connesso agli altri nodi per il tramite di archi orientati

- Un arco uscente da un modulo M e diretto verso un modulo A rappresenta la relazione M USA A



- I nodi riportano la struttura interna dei moduli: i moduli P, Q e R sono componenti del modulo M

- Struttura interna di M:

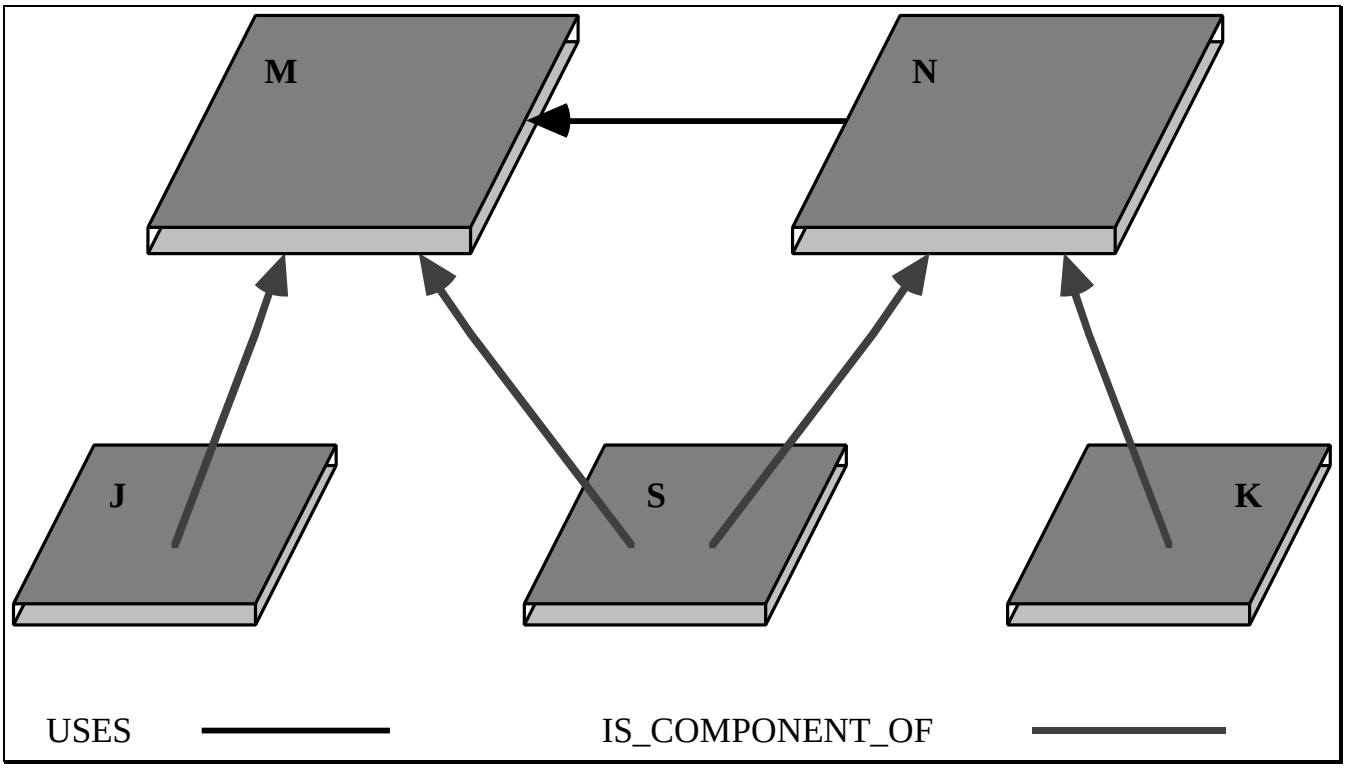


- Quando uno stesso modulo e` un componente di due o più moduli:

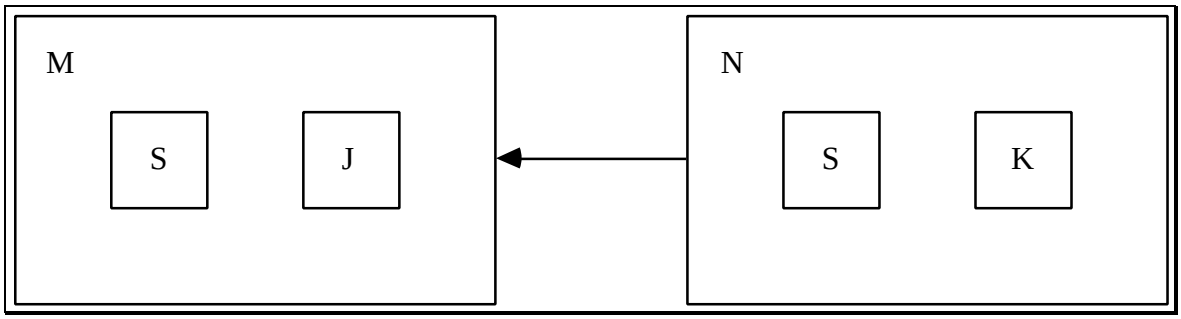
  (1) si rappresenta tale modulo come indipendente e si introducono archi diversi per le relazioni USA e COMPONENTE_DI

Esempio

Si supponga che uno stesso modulo S sia componente di un modulo M, che contiene anche un modulo J, e di un modulo N, che contiene anche un modulo K; il modulo N sia inoltre in relazione USA con il modulo M



(2) oppure si replica il modulo all'interno di tutti i moduli di cui è componente



I moduli come astrazione sul controllo: Librerie

- Nei linguaggi di programmazione tradizionali per modulo si intende una prima forma di *astrazione* effettuata sul *flusso di controllo*

- Il concetto di modulo è identificato con il concetto di *subroutine* o *procedura*

- Una procedura può effettivamente nascondere una scelta di progetto riguardante l'algoritmo utilizzato

  Ad esempio, per l'ordinamento di un vettore di elementi interi si hanno più algoritmi diversi per tale compito: bubblesort, shellsort, quicksort, etc.

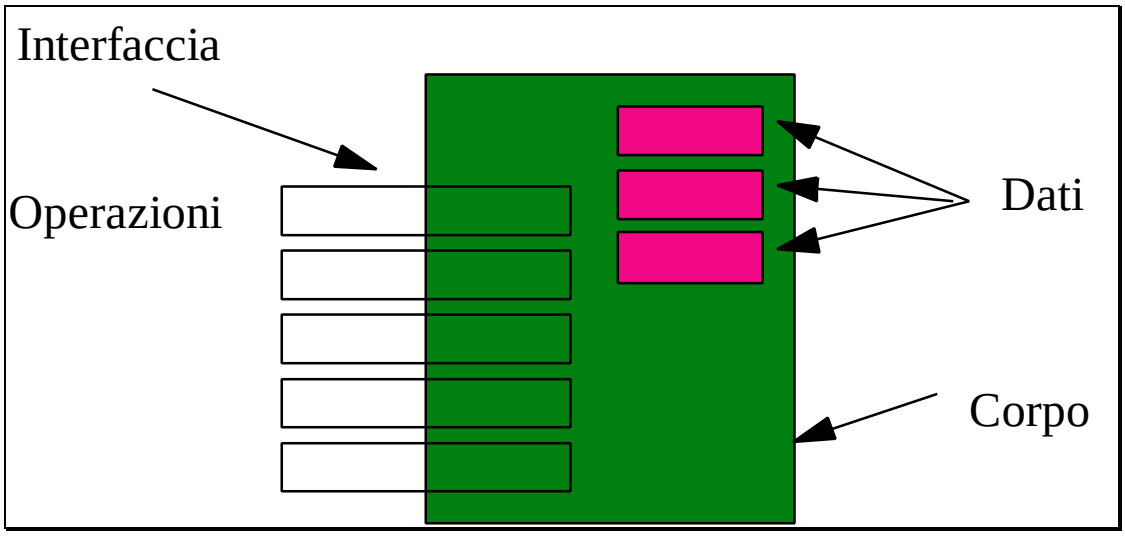
- Il principio di information hiding puo` essere disatteso qualora compaiano effetti collaterali

- Le procedure in quanto astrazioni sul controllo sono utilizzate come parti di alcune classi di moduli, che prendono il nome di *librerie* (ad esempio, librerie di funzioni matematiche e grafiche)

I moduli come Astrazioni di Dato

- Un'Astrazione di Dato (o Struttura Dati Astratta, ADS) è un modo di *incapsulare* un *dato* in una rappresentazione tale da regolamentarne l'accesso e la modifica

- *Interfaccia* dell'oggetto (operazioni) *stabile* anche in presenza di modifiche alla struttura dati



- Risultati dell'attivazione di un'operazione su un oggetto dipendenti anche dal valore del dato (*stato*)
- Non puramente funzionali (come le *librerie*)

Definizione TDN di un archivio anagrafico:

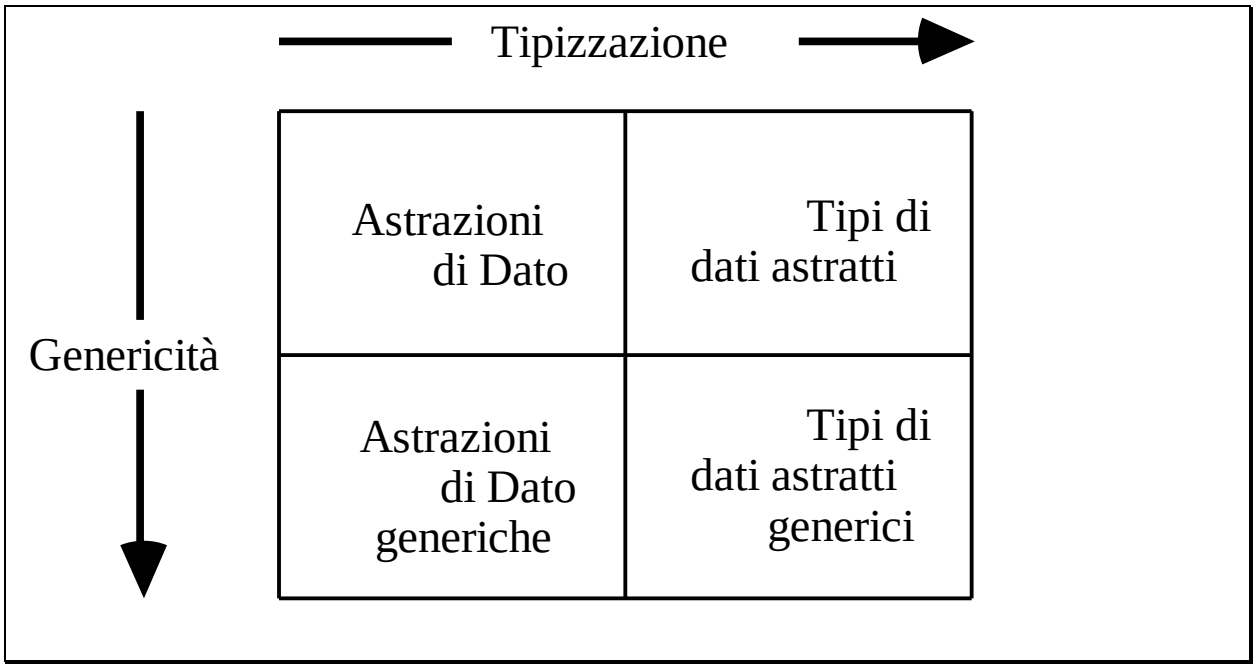
```
module ArchivioAnagrafico;
uses …
exports
    function NumeroPersone(): Integer;
    procedure Inserisci(in P: Persona);
    function Esiste(in P: Persona): Boolean;
    procedure Elimina(in P: Persona);
    procedure Inserisci(in P: Persona);
    …
implementation
    …
end ArchivioAnagrafico;
```

Moduli come astrazioni sui dati

- L'astrazione sui dati può procedere in due direzioni complementari:

  - il passaggio da astrazioni di dato a tipi di dati astratti

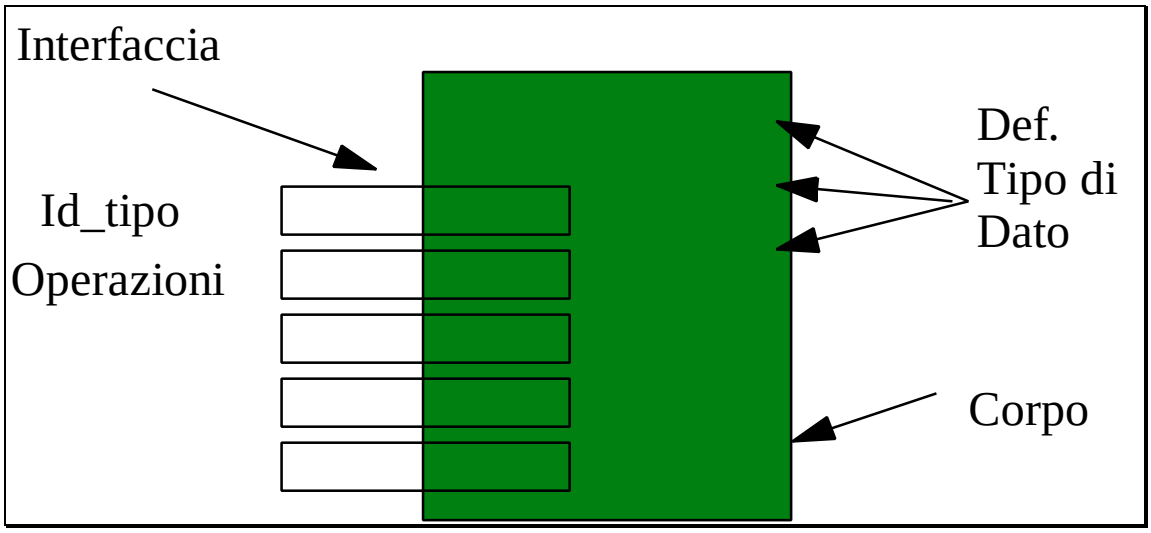
  - il passaggio da astrazioni/tipi di dato ad astrazioni/tipi generici



Tipi di dati astratti

- Un Tipo di dato Astratto (ADT) descrive la struttura interna degli oggetti che fanno riferimento al tipo e l'insieme di risorse che gli oggetti del tipo in esame esportano

- La struttura del dato risulta invisibile all'esterno



Esempio TDN di un ADT:

```
module NumeriComplessi;

uses …

exports

    type Complesso = ?;

    function Somma(in Addendo1, Addendo2: Complesso): Complesso;

    function Sottrazione(in Addendo1, Addendo2: Complesso): Complesso;

    function Moltiplicazione(in Addendo1,Addendo2:Complesso):
    Complesso;

    function Divisione(in Addendo1, Addendo2: Complesso): Complesso;

    function ParteReale(in Numero: Complesso): Real;

    function ParteImmaginaria(in Numero: Complesso): Real;

    function Modulo(in Numero: Complesso): Real;

    function Argomento(in Numero: Complesso): Real;

    function Inizializzazione(in ParteReale,ParteIm: Complesso):
    Complesso;

…

end NumeriComplessi;
```

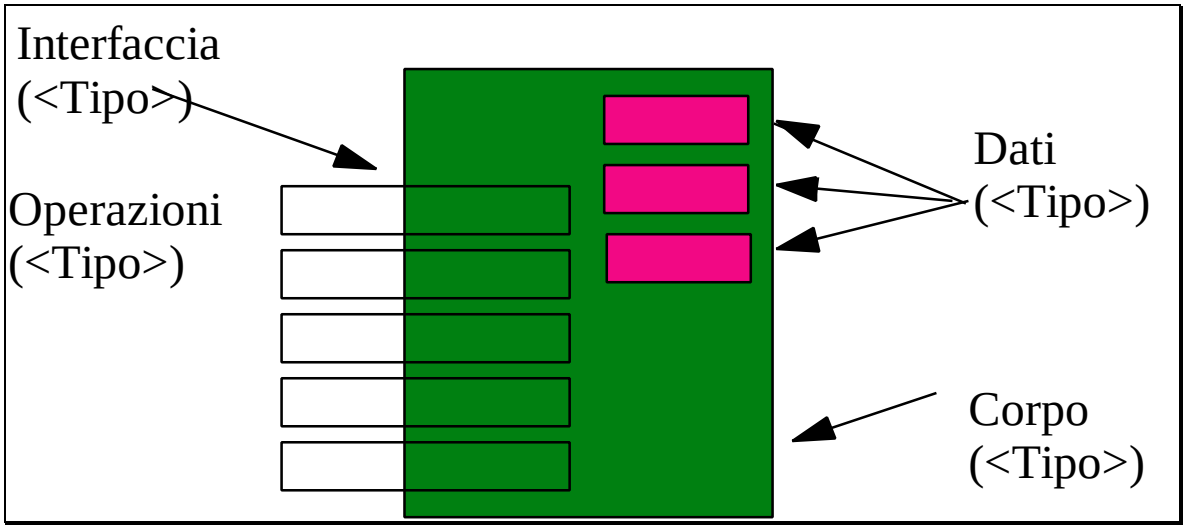
- Il modulo esporta il tipo *Complesso*, ma non rende visibili in alcun modo i dettagli realizzativi

Oggetti generici

- Spesso le modalità di manipolazione di un oggetto per così dire "composto" non dipendono dall'effettiva natura dei suoi "componenti"

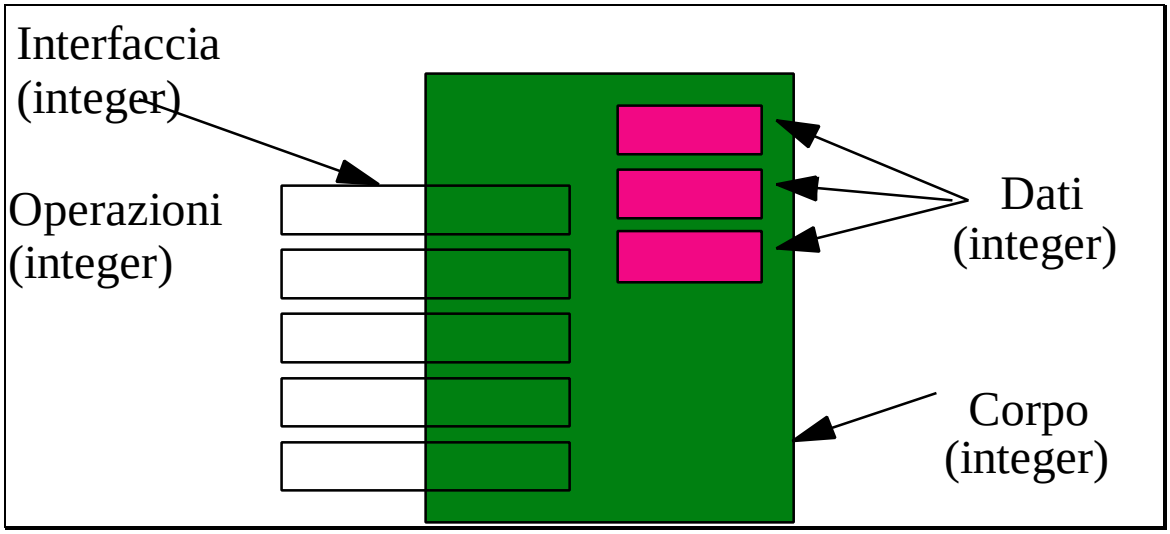
- Ad esempio, tabella di interi o reali: le operazioni sono indipendenti dal tipo dei dati

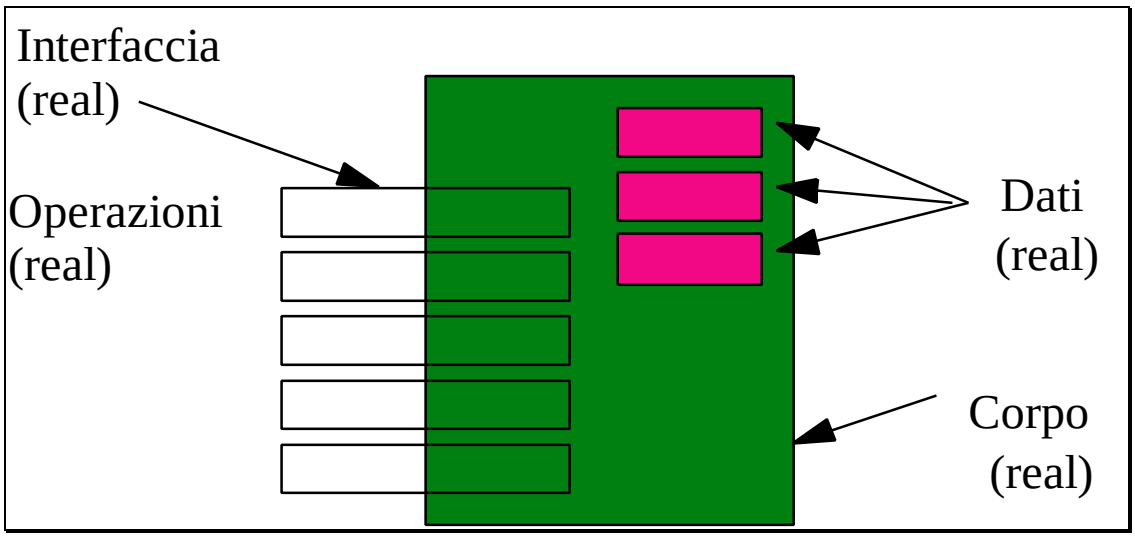
- Un oggetto generico è perciò, in un certo senso, *parametrico* rispetto ad un *tipo* (ad esempio, tipo dei suoi componenti)



- Rappresenta lo *schema* (in inglese *template*) da cui vengono ricavati gli oggetti (astrazioni di dato, in questo caso)

- Definizione di un'istanza specificando il parametro *tipo*





Definizione TDN di tabella generica

```
generic module TabellaGenerica(TipoComponente);

uses …

exports

    procedure Aggiungi(in Componente: TipoComponente);

    procedure Elimina(in Componente: TipoComponente);

    procedure Modifica(in VecchioComponente,NuovoComponente:
                                                    TipoComponente);

    function Esiste(in Componente: TipoComponente): Boolean;

    function Ricerca(in Componente:TipoComponente):
                                                    TipoComponente;

implementation

…

end TabellaGenerica;
```

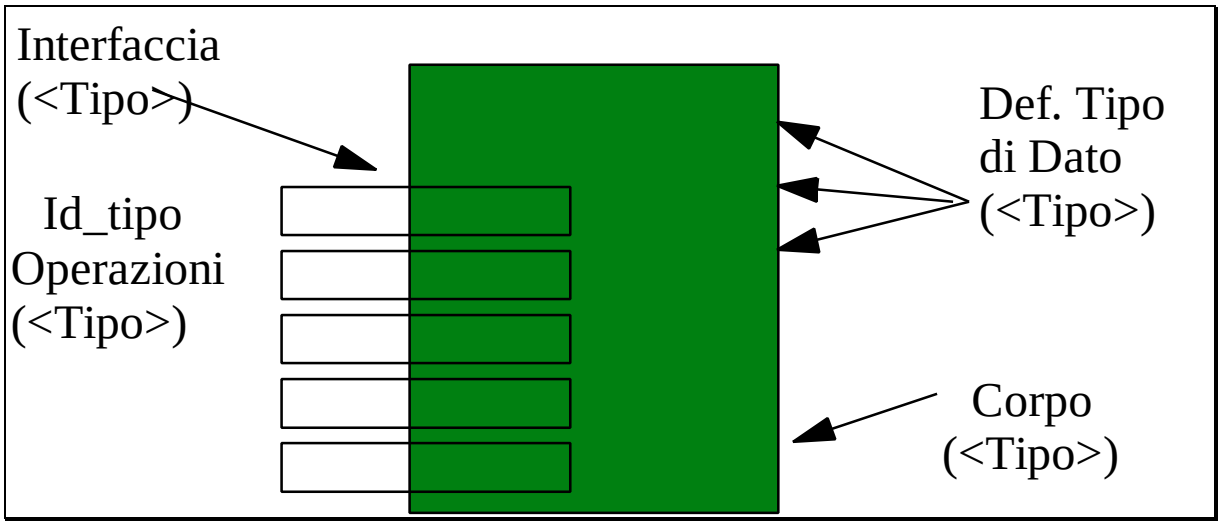
- Il modulo, generico rispetto al tipo TipoComponente, prima di essere usato deve essere *istanziato* (definizione del tipo dei componenti)

```
…

module TabellaDiInteri is TabellaGenerica(Integer);

module TabellaDiNomi is TabellaGenerica(Persona);
```

...

Tipi di dati astratti generici

- Tipi parametrici rispetto alla natura delle loro componenti



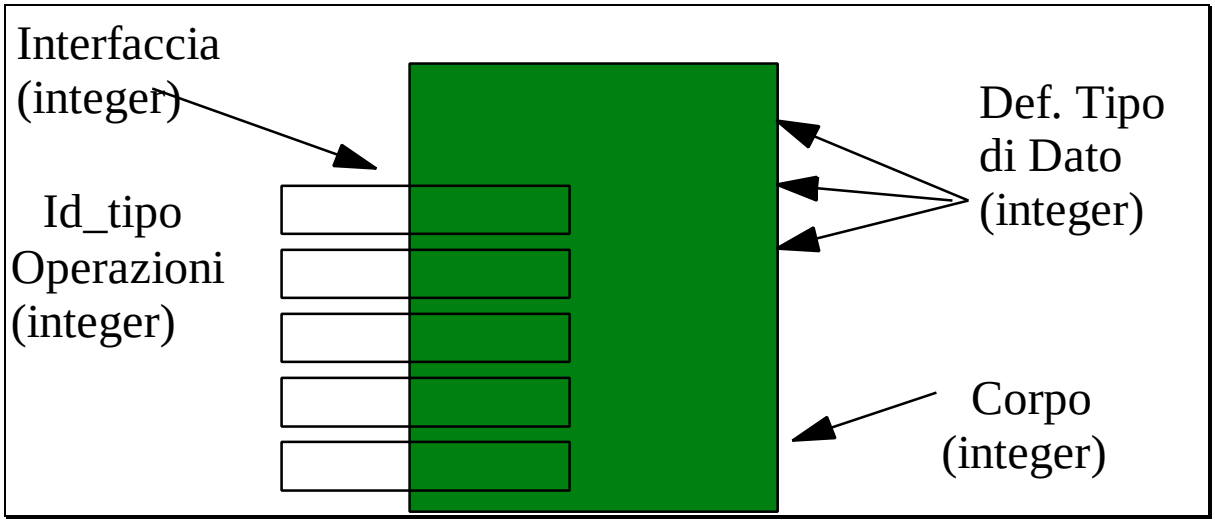
- Ad esempio il tipo di dato astratto Insieme, che realizza il concetto di insieme matematico

Definizione TDN di ADT generico

```
generic module InsiemeGenerico(TipoElemento);

uses …

exports

    type TipoInsieme = ?;

    function Unione (in Insieme1, Insieme2: TipoInsieme):
                                                    TipoInsieme;

    function Intersezione (in Insieme1, Insieme2: TipoInsieme):
                                                    TipoInsieme;

    function Complementazione (in Insieme1, Insieme2: TipoInsieme):
                                                    TipoInsieme;

    function Is_Vuoto(in Insieme: TipoInsieme): Boolean;

    function Esiste (    in Insieme1: TipoInsieme;
                         in Elemento: TipoElemento): Boolean;

    procedure Aggiungi (     inout Insieme1: TipoInsieme;
                             in Elemento: TipoElemento);

    procedure Elimina (inout Insieme1: TipoInsieme;
                        in Elemento: TipoElemento);

    function Vuoto: TipoInsieme;

implementation

…

end InsiemeGenerico;
```

Definizione di una istanza

- Per usare un ADT generico occorre *istanziarlo*
- Costruzione di un ADT specificando un tipo effettivo al posto del tipo generico




```
module InsiemeDiInteri is InsiemeGenerico(Integer);
```

- Sia le astrazioni di dato che i tipi di dato astratto possono poi essere generici anche rispetto a procedure

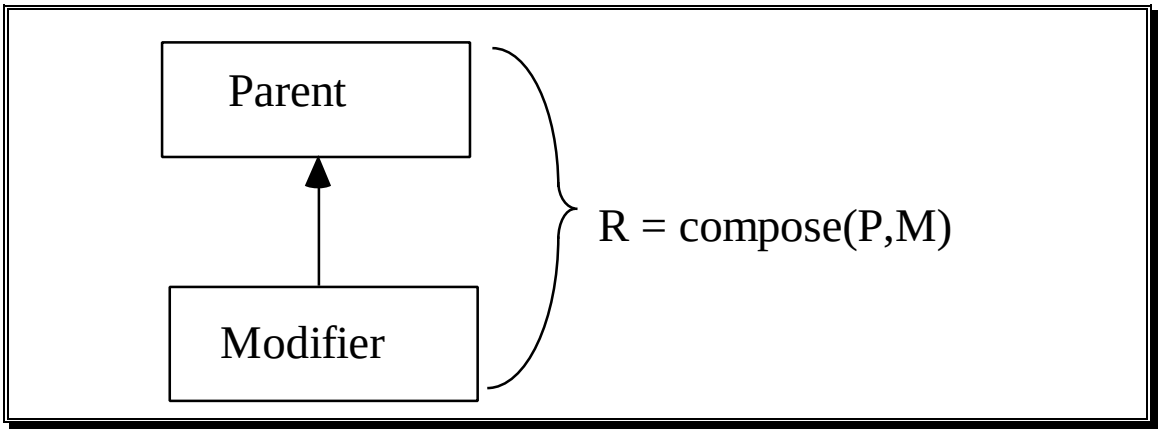
Progettazione orientata agli oggetti

- Favorisce la riusabilità dei componenti, la loro affidabilità ed un procedimento di sviluppo incrementale del prodotto software

- Alle caratteristiche di *information hiding* viene aggiunta la possibilità di definire un modulo come *specializzazione* di un altro modulo con l'aggiunta di ulteriori proprietà

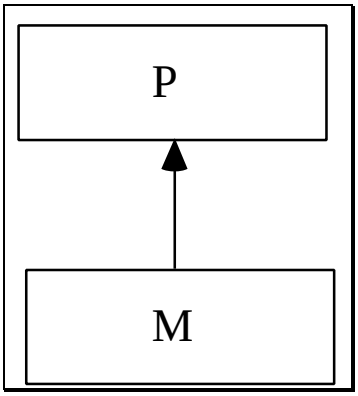
- Ulteriore relazione, EREDITA_DA, detta di ***ereditarietà*** (anche IS_A)

- L'ereditarieta` consente di costruire un modulo software dato da un Parent (*genitore*) piu` un Modifier (*modificatore*):



Ereditarieta`

- M IS_A P

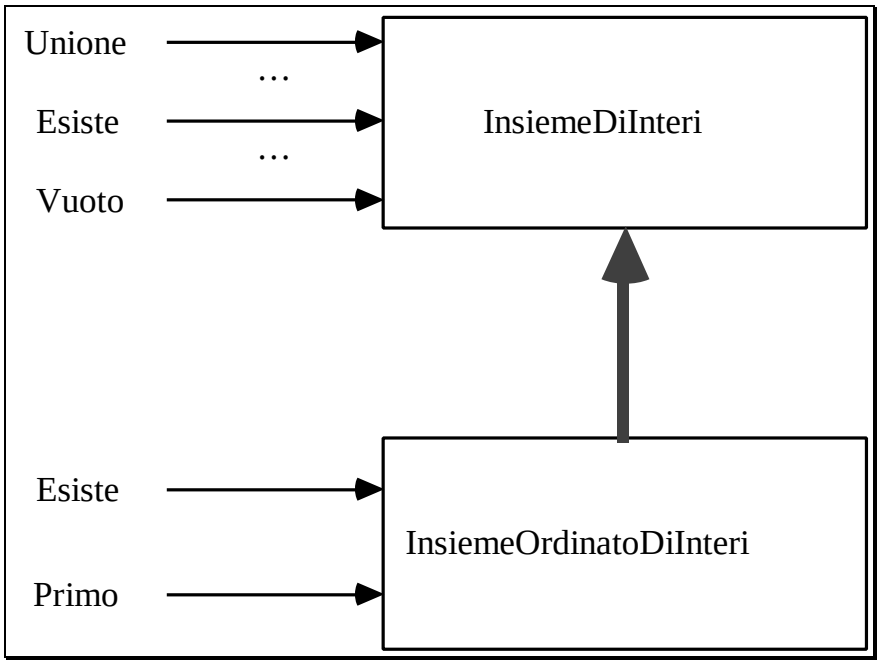


  - P, modulo *genitore*
  - M, modulo *erede* (o figlio)

- Il modulo M *eredita* perciò tutte le caratteristiche di P, ossia mette anch'esso a disposizione tutte le risorse esportate da P
- Alcune di tali risorse possono però essere ridefinite (***overriding***)
- Nel modulo erede possono essere definite ulteriori risorse esportate (***raffinamento***)

Esempio:

- InsiemeOrdinatoDiInteri come una specializzazione di InsiemeDiInteri (aggiunge la funzione PrimoElemento)




```
module InsiemeOrdinatoDiInteri inherits InsiemeDiInteri;
…
exports
    function PrimoElemento(in I: InsiemeOrdinatoDiInteri): Integer;
    function Esiste(in I: InsiemeOrdinatoDiInteri; in N: Integer): Integer;
…
end InsiemeOrdinatoDiInteri;
```