Linguaggi

Corso di Laurea in Ingegneria delle Telecomunicazioni
A.A. 2010-2011

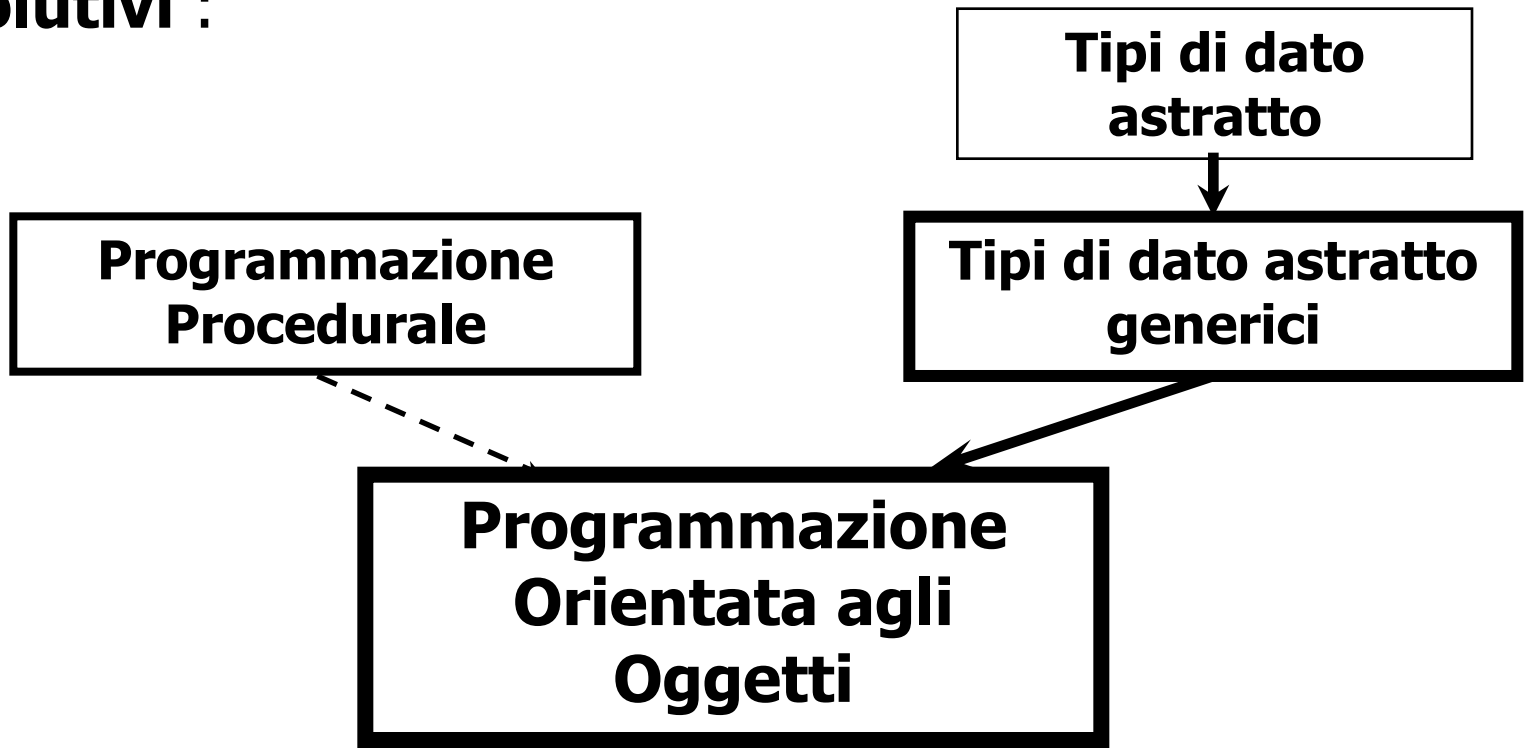
Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

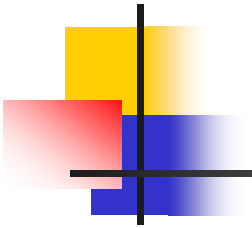
- lezione 02 -

Introduzione alla programmazione Object-Oriented

Programmazione Object-Oriented

La programmazione orientata agli oggetti rappresenta il **punto di convergenza** di due differenti **percorsi evolutivi** :

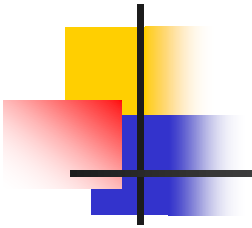




Programmazione procedurale

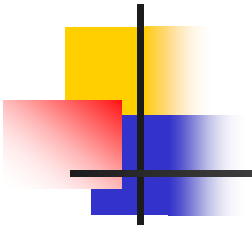
La programmazione procedurale presenta alcuni **limiti** :

- È richiesta la **creazione e distruzione esplicita** delle strutture dati utilizzate.
- Esiste uno **stretto legame tra dati e operazioni sui dati** (non vi è una netta e chiara separazione fra i due). Questo comporta:
  - La mancanza di sicurezza dei dati (non averli cioè nettamente separati comporta il rischio che su essi possano essere effettuate operazioni non desiderate).
  - Un forte legame tra rappresentazione dei dati e strategia di visita degli stessi.



Tipi di dato astratti (ADT)

- La caratteristica fondamentale dell'ADT è l'encapsulation, secondo cui si racchiude tutto l'ADT (ossia la specifica del tipo di dato e delle operazioni effettuabili su di esso) in un unico modulo. Dunque gli ADT:
  - Esportano un tipo di dato.
  - Esportano un insieme di operazioni (interfaccia).
  - Prevedono che le operazioni esportate siano il solo meccanismo per manipolare i dati.
  - Contengono assiomi e precondizioni che definiscono il dominio di applicazioni del tipo di dato.
- Un **ADT generico** è una modalità di utilizzo di un ADT secondo la quale si definisce un nuovo tipo ad ogni instanziazione dell'ADT sulla base dei valori assunti da alcuni parametri (che quindi permettono la generalizzazione)



Generalità programmazione OO

La programmazione object-oriented sfrutta le innovazioni concettuali dei tipi di dato astratto generici per superare i limiti della programmazione procedurale, offrendo un

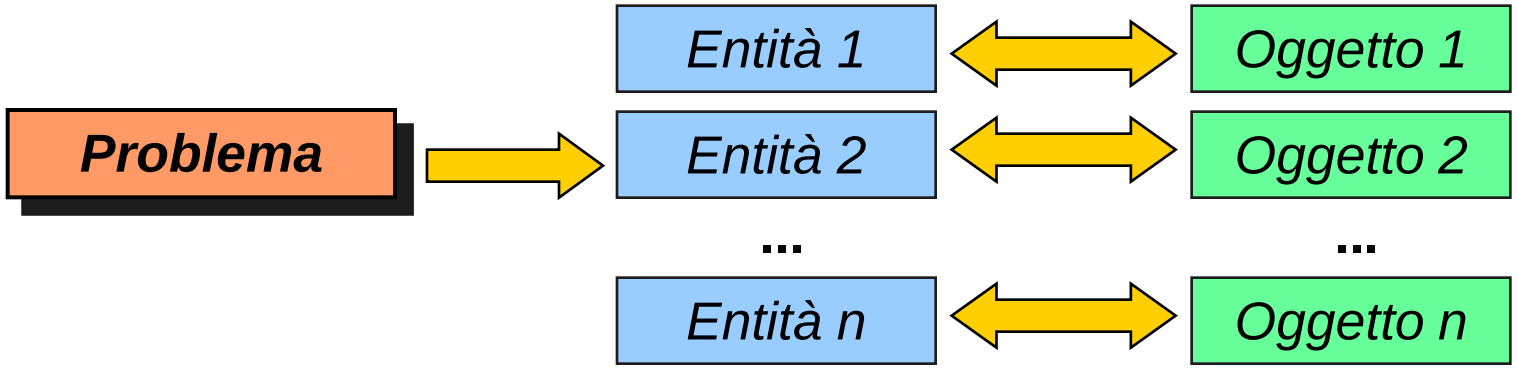
Nuovo paradigma di programmazione

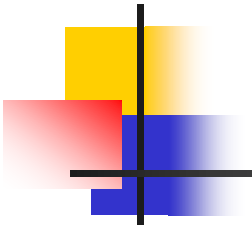
La programmazione ad oggetti rappresenta in tal senso un modo diverso di pensare alla risoluzione dei problemi.

Invece di modellare il problema adattandolo alla logica del calcolatore infatti, l'approccio orientato agli oggetti **adatta il calcolatore al problema.**

Generalità programmazione OO

Il problema da risolvere viene analizzato individuando delle **entità indipendenti** che sono in relazione con le altre. Tali entità non vengono scelte perché facilmente traducibili nella logica di programmazione ma in quanto esistenti nel problema in esame e caratterizzate da vincoli che le isolano o le evidenziano dal contesto del problema stesso.
Tali entità sono rappresentate come **oggetti del programma**: lo scopo è di ottenere una corrispondenza biunivoca tra entità del problema e oggetti del programma. I progettisti possono quindi concentrarsi sui dati che stanno alla base del problema anziché sulle funzionalità del programma.





Oggetti

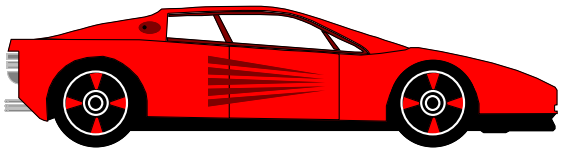
- Gli Oggetti sono cose
- Gli Oggetti possono essere semplici o complessi
- Gli Oggetti possono essere reali o immaginari
- Qualsiasi cosa può essere considerata un oggetto.
- Generalmente si considerano come oggetti tutte quelle entità dotate di una propria identità.
- Non tutte le entità di un problema è detto che debbano essere modellate come oggetti, dipende dalla natura del problema, che può consentire o richiedere di trascurare o meno certe entità perché non significative per il problema stesso.
- Possono esistere oggetti composti da altri oggetti. In tal caso, si deve approfondire l'analisi ed individuare i singoli componenti sino a che il livello più basso contenga soltanto oggetti atomici.

Oggetti

Alcune cose non sono oggetti ma sono **attributi,** valori o caratteristiche di oggetti (velocità, dimensioni, ecc.).
A seconda della natura del problema, una stessa entità può dover essere rappresentata talora come oggetto, talora come attributo.
In questo secondo caso, occorre determinare anche l'oggetto a cui apparterebbe (un attributo deve essere sempre associato a qualche oggetto). Un'entità non può essere contemporaneamente attributo ed oggetto.

oggetto → automobile

Esempio



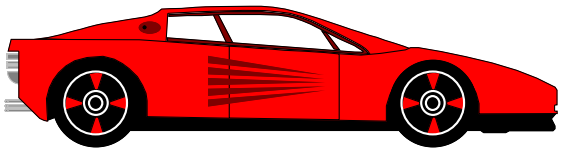
attributi → Velocità, colore, prezzo, dimensioni

Oggetti

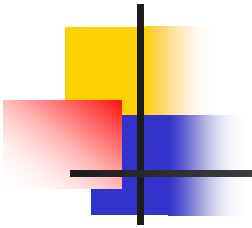
Oltre gli attributi**,** un oggetto è caratterizzato dalle azioni che nel mondo reale esso può fare e/o subire**.** Tali azioni si realizzano mediante funzioni (note anche con il nome di **metodi**), che spesso effettuano al loro interno operazioni che coinvolgono gli attributi dell'oggetto stesso**.**

Esempio

oggetto → automobile



metodi → Girare, muoversi, fermarsi, tamponare

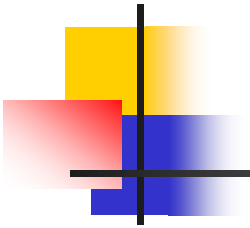


Oggetti

Per definire un oggetto è quindi necessario individuare:

- Le sue caratteristiche (attributi), effettuando la cosiddetta **astrazione sui dati**
- Le azioni che può fare e/o subire (metodi), effettuando la cosiddetta **astrazione funzionale**

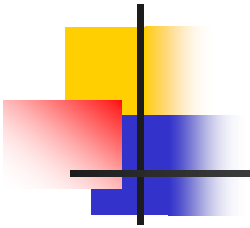
Nella fase di progettazione, non occorre specificare come verranno implementati i metodi e gli attributi.



Oggetti

Oggetto = auto

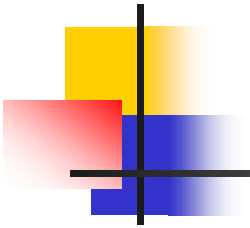
- Le cose che un'auto può fare sono:
  - Andare
  - Fermarsi
  - Girare a destra
  - Girare a sinistra
- Noi sappiamo che l'auto fa queste cose, ma non conosciamo come vengono fatte, cioè non sappiamo come vengono implementate.



Oggetti

Oggetto = auto

- Un'auto può avere le seguenti caratteristiche o attributi:
  - Colore
  - Velocità max
  - Dimensioni
  - Tipo di carburante
- Come questi attributi sono memorizzati o definiti è irrilevante per il progetto dell'oggetto.



Classi

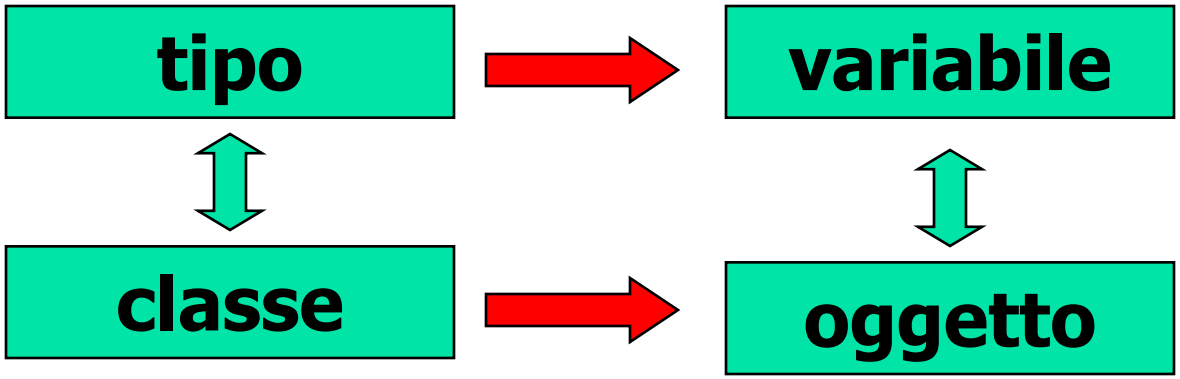
Una **classe** rappresenta l'insieme di tutti quegli oggetti che condividono le stesse caratteristiche in termini di struttura (non di valore posseduto).

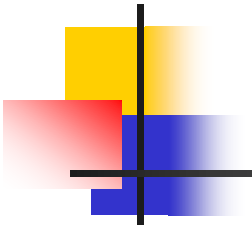
Classe	Autovettura
BMW520	blu
BMW520	rossa

due autovetture aventi lo stesso attributo ma con diversi colori (valori)
Due oggetti della stessa classe possono anche avere valori identici degli attributi; restano sempre e comunque distinti (occuperanno zone diverse della memoria)

Classi

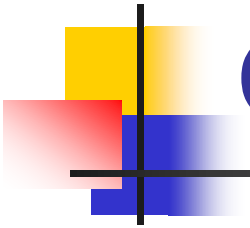
- "classe" è quindi un sinonimo di "tipo" dei linguaggi procedurali, ma introduce un significativo livello di astrazione in più
- Un oggetto incarna l'idea di ADT in senso completo: viene creato dalla classe ed è capace di esportare operazioni (metodi), denominate anche **interfaccia**, ed incapsulare lo stato interno (rappresentato dai valori degli attributi)





Classi

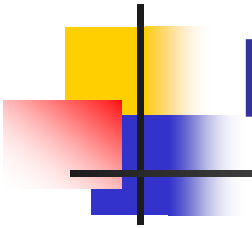
- Una classe viene creata prima dei suoi oggetti.
- L'approccio NON è quello di creare in qualche modo i vari oggetti e successivamente raggrupparli in base alle proprietà possedute formando le classi.
- Dopo avere definito la classe, si può quindi creare un suo oggetto, azione denominata **istanziazione** (corrisponde, nei linguaggi procedurali, a creare una variabile di un certo tipo).
- Ogni istanza di una classe è un oggetto distinto ed indipendente dagli altri oggetti della stessa classe (diverse zone di memoria), quindi l'esecuzione di un metodo, o l'alterazione del valore di un qualche attributo, pur essendo metodi ed attributi specificati entro la classe, sono azioni che hanno **effetto solo sulla singola istanza** su cui sono invocati.



Caratteristiche Linguaggi OO

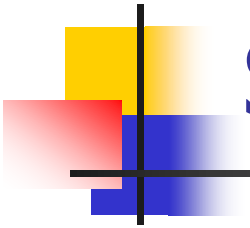
I linguaggi OO hanno le seguenti caratteristiche:

- Message passing
- System hiding
- Ereditarieta
- Polimorfismo



Caratteristiche Linguaggi OO
Message passing

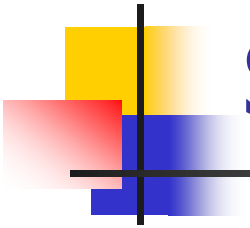
Secondo il concetto di **message passing**, un oggetto interagisce con gli altri mandando messaggi.
L'invio del messaggio corrisponde all'invocazione, da parte dell'oggetto chiamante, del metodo dell'oggetto chiamato.
Contestualmente, si possono inviare e/o ricevere parametri.
L'oggetto chiamato può rispondere (reply), cambiare il proprio stato o reagire in qualsiasi altro modo.
Gli oggetti possono interagire solo attraverso la loro interfaccia pubblica (metodi pubblici).



Caratteristiche Linguaggi OO
System hiding

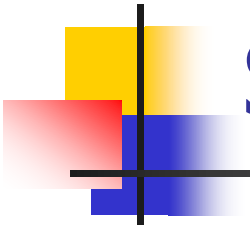
I metodi, gli attributi o anche una stessa classe possono essere **pubblici o privati**:

- un metodo **privato** è inaccessibile dall'esterno. Fornisce servizi che il creatore della classe non vuole rendere direttamente accessibili
- Un metodo **pubblico** è invece utilizzabile dall'esterno.
- Un attributo **privato** è una variabile interna, manipolata da metodi privati o pubblici che siano, ma comunque non *direttamente* modificabile dall'esterno.
- Un attributo **pubblico** invece può essere modificato dall'esterno, senza passare attraverso un metodo. È in genere comunque sconsigliabile agire su tutti gli attributi tramite variabili, se non è strettamente necessario, questo perche' un metodo può fare da filtro, ad esempio impedendo di scrivere valori non ammessi nell'attributo.



Caratteristiche Linguaggi OO System hiding

- una **classe** privata è una classe inaccessibile dall'esterno, e che sarà quindi utilizzata da una qualche altra classe, privata o meno.
- Questo discorso ha chiaramente senso nel contesto di un gruppo di classi in cui almeno una sia pubblica, in modo che l'utente possa usare il pacchetto (package) di classi
- Il fatto che tutto ciò che non è specificato come pubblico non è accessibile all'esterno garantisce che nessuno possa effettuare usi impropri di attributi, metodi o classi, prevenendo così il verificarsi di eventuali condizioni di errore.
- Tale caratteristica è nota come **system hiding**

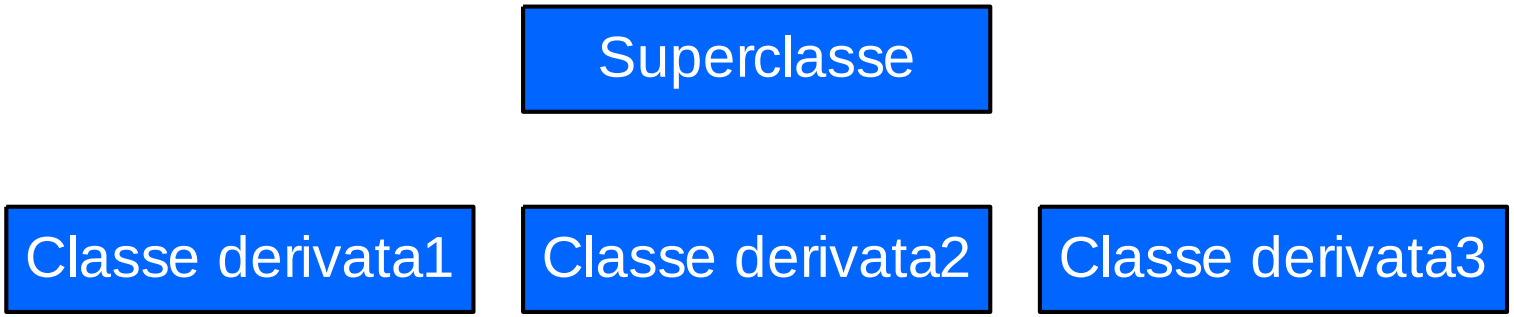


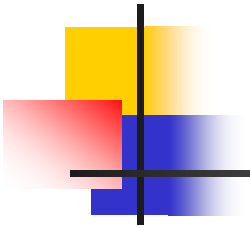
Caratteristiche Linguaggi OO
System hiding

- Il system hiding dei metodi offre anche un altro valore aggiunto, quello del **disaccoppiamento specifica - implementazione**.
- Esiste infatti la possibilità di cambiare la parte privata, adattando di conseguenza il codice interno dei metodi pubblici che la usano, il tutto senza che il codice che utilizzava quella classe avverta la variazione effettuata (e quindi esso non richiede cambiamenti), questo poiché esso utilizza solo metodi pubblici, i cui **prototipi** (o **signature**, ossia il nome del metodo ed i suoi parametri di I/O) non sono stati cambiati.

Caratteristiche Linguaggi OO
Ereditarietà

Una classe può essere creata completamente da zero, oppure, se serve, può ereditare le caratteristiche di un'altra (che sarà denominata **superclasse** della classe appena creata, detta **derivata**).



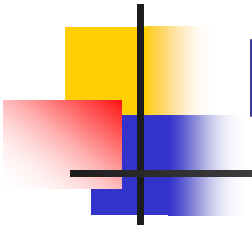


Caratteristiche Linguaggi OO Ereditarietà

Chiaramente, si presume che la classe derivata abbia qualche differenza con la superclasse. Tale differenza può concretizzarsi:

- nell'avere caratteristiche aggiuntive (attributi e/o metodi);
- nel variare qualche caratteristica presente nella superclasse, effettuando il cosiddetto overriding.

Tale variazione non ha ripercussioni nella superclasse.



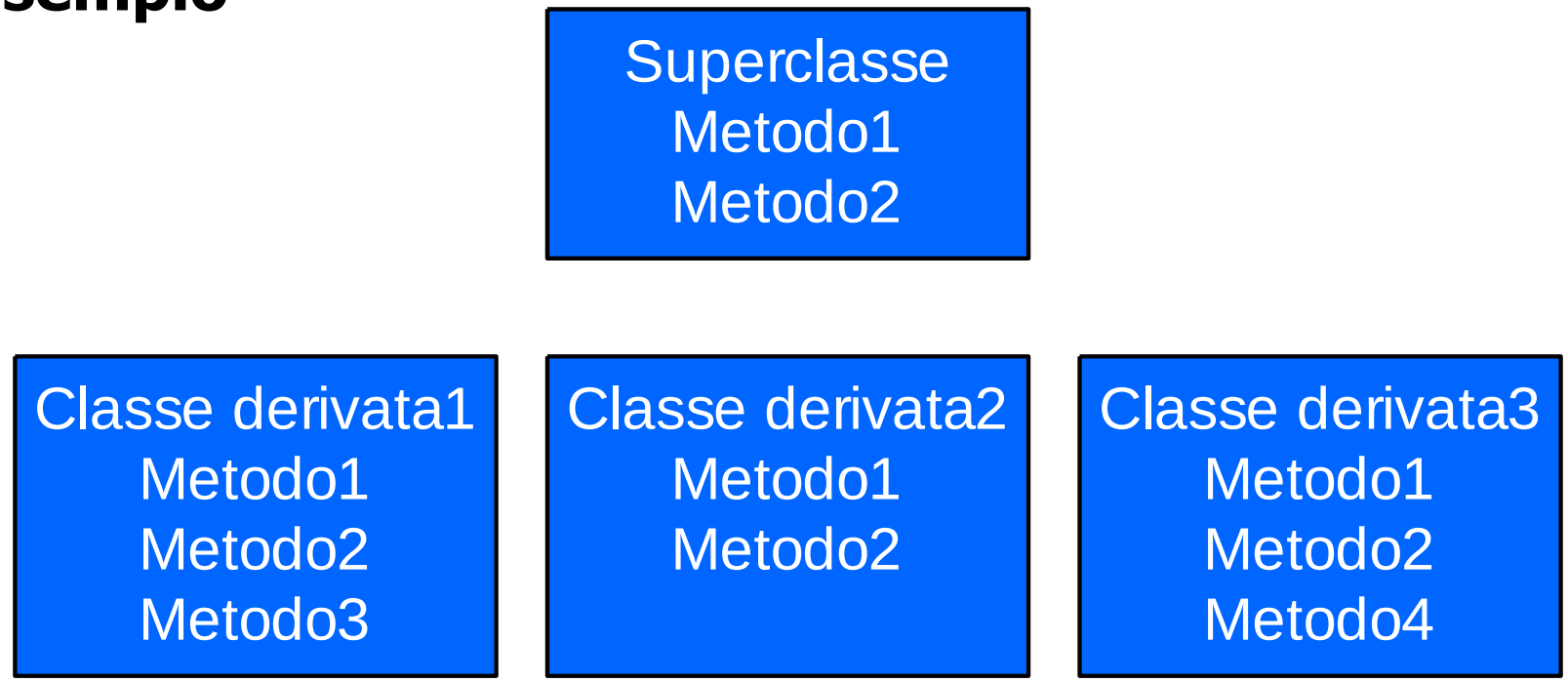
Caratteristiche Linguaggi OO Ereditarietà

Si osserva che un qualunque cambiamento effettuato nella superclasse, modifica automaticamente le classi derivate, comprese le loro istanze.

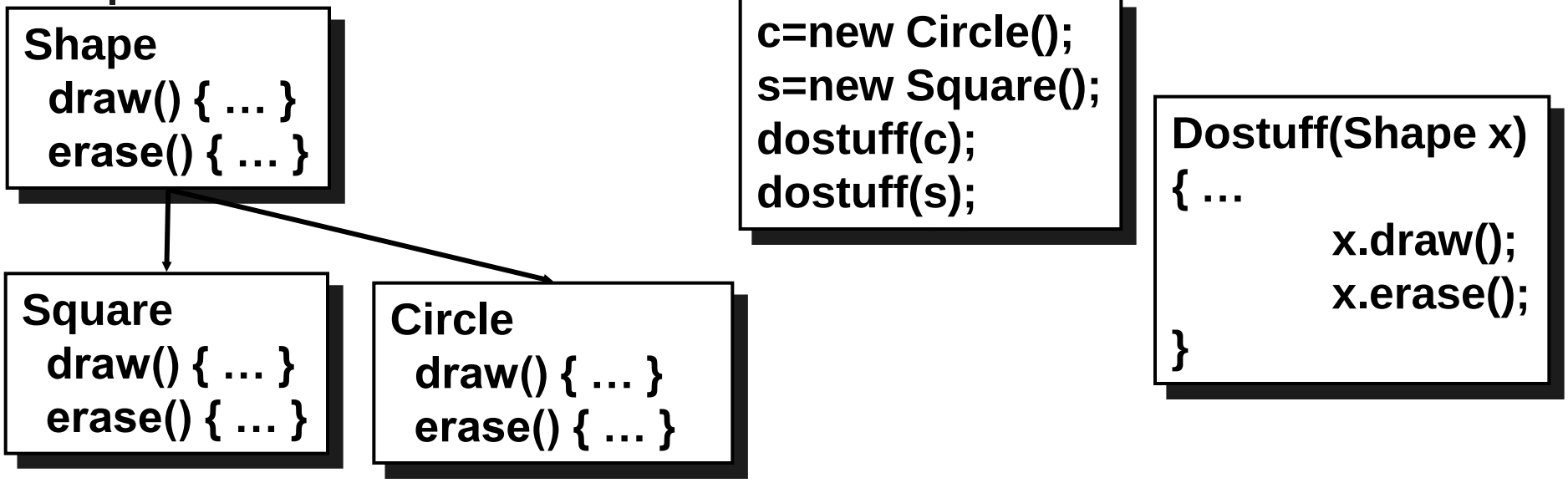
Una classe derivata non può avere meno caratteristiche della sua superclasse, in quanto il concetto guida dell'ereditarietà è che dovunque si usi un oggetto della superclasse, deve essere possibile sostituirlo con un oggetto di una qualunque delle classi derivate.

Caratteristiche Linguaggi OO
Ereditarietà

Esempio

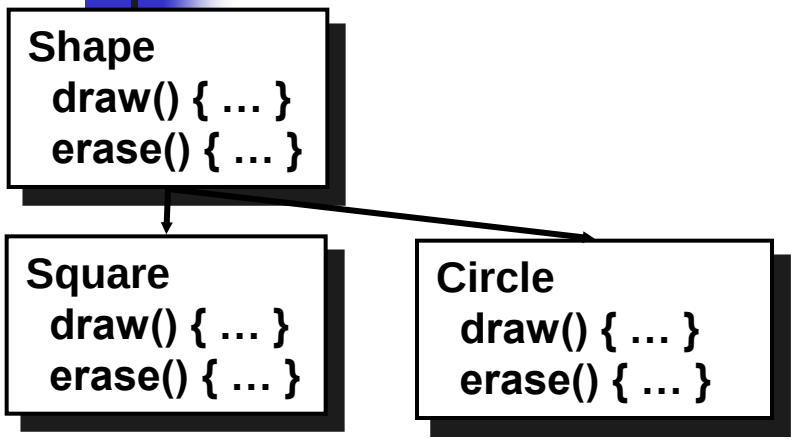


Caratteristiche Linguaggi OO
Polimorfismo



Durante l'esecuzione del codice, anzitutto c ed s sono riconosciuti ad accettati come validi in quanto oggetti istanze di classi derivate entrambe da Shape, pertanto verranno trattati come se fossero istanze della superclasse (***upcasting***), anche se essi potranno sfruttare le proprie caratteristiche aggiuntive.

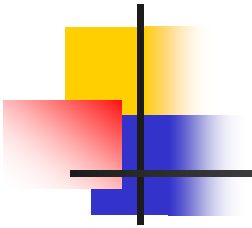
Caratteristiche Linguaggi OO Polimorfismo




```
c=new Circle();
s=new Square();
dostuff(c);
dostuff(s);
```

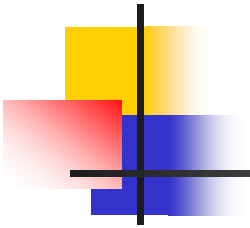
```
Dostuff(Shape x)
{ …
        x.draw();
        x.erase();
}
```

Successivamente, i metodi draw() ed erase() invocati dentro il metodo dostuff vengono associati ai corrispettivi alter-ego di Circle (per dostuff(c)) e di Square (per dostuff(s)). Questo meccanismo è il ***dynamic binding***.
In tal modo, il codice di dostuff si adatta alla classe a cui appartiene il parametro x correntemente passato, nel senso che per dostuff(c) verrà usato il metodo draw della classe Circle, mentre quando viene invocato dostuff(s), verranno eseguiti sempre i metodi draw ed erase, ma relativi alla classe Square.
Le due coppie di metodi, pur avendo gli stessi nomi in dostuff, potrebbero essere implementate in maniera completamente diversa. Questa capacità del linguaggio di consentire un adattamento del codice è il ***polimorfismo***.



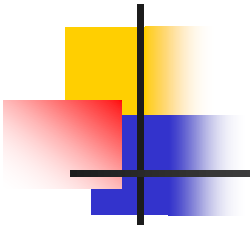
Vantaggi Linguaggi OO

- Consente di implementare gli oggetti come semplice immediata ed intuitiva rappresentazione del mondo reale.
- Assicura che l'implementazione degli oggetti possa essere nascosta al programmatore.
- Permette di raggiungere un elevato livello di reimpiego del software.
- Limitazione dell'accesso ai dati.
- Riduzione del tempo e del costo di sviluppo:
  - utilizzo di componenti di software riutilizzabili come classi basi.
  - utilizzo della risoluzione incrementale dei problemi grazie alle sottoclassi.



Vantaggi Linguaggi OO

- Affidabilita':
  - può essere aumentata grazie all'elevato livello di integrazione insisito nella progettazione del sistema.
  - la verifica dell'integrazione può essere effettuata ad alto livello.
- Rapida realizzazione dei prototipi:
  - una volta completata la scomposizione del sistema in classi e oggetti appartenenti alle classi molti dei metodi principali caratterizzanti possono essere implementati velocemente tramite le sottoclassi.
- Riduzione del costo di manutenzione:
  - modifica di una o più classi.



Qualche Termine

- **contratto** di una classe
  - insieme di dati e metodi accessibili (visibili) e della relativa semantica
- **interfaccia** di una classe
  - elenco dei metodi e dei dati accessibili
- **interfaccia** (**signature**) di un metodo
  - nome del metodo più eventuali parametri