Introduzione al modello COM

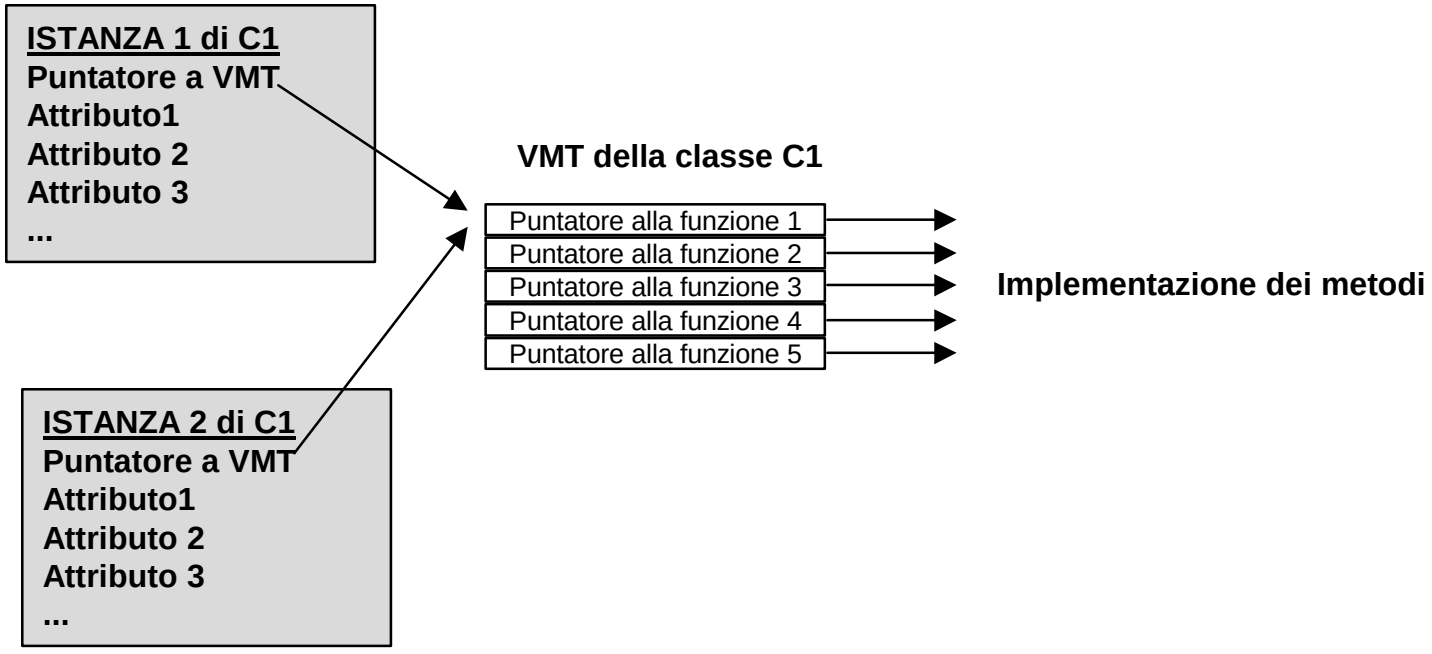
Enrico Lodolo
e.lodolo@bo.nettuno.it

Metodi virtuali

- I metodi virtuali sono i metodi che possono essere ridefiniti nelle classi derivate
- Consentono l'implementazione del polimorfismo
- In Java i metodi sono per default virtuali, a meno che non vengano identificati come final
- In C++ e in Object Pascal invece i metodi sono per default statici (non virtuali), a meno che non vengano identificati come virtual
- Le chiamate a metodi statici vengono risolte completamente a tempo di compilazione (early binding). In pratica abbiamo un jump ad un indirizzo preciso.
- Le chiamate a metodi virtuali vengono invece risolte a tempo di esecuzione (late binding). La chiamata produce un jump indiretto e quindi c'è un leggero overhead.
- Nei linguaggi ad oggetti compilati i metodi virtuali vengono implementati per mezzo della "virtual method table"

Virtual method table

- **Una virtual method table (VMT o vtable) è una tabella di puntatori a funzione**
- **Esiste una VMT per ogni classe**
- **Un'istanza è una struttura dati che contiene lo stato di un oggetto**
- **Il primo campo di questa struttura dati è il puntatore alla VMT della classe a cui l'oggetto appartiene**
- **Tutte le istanze di una classe puntano alla stessa VMT**



VMT, late binding e polimorfismo

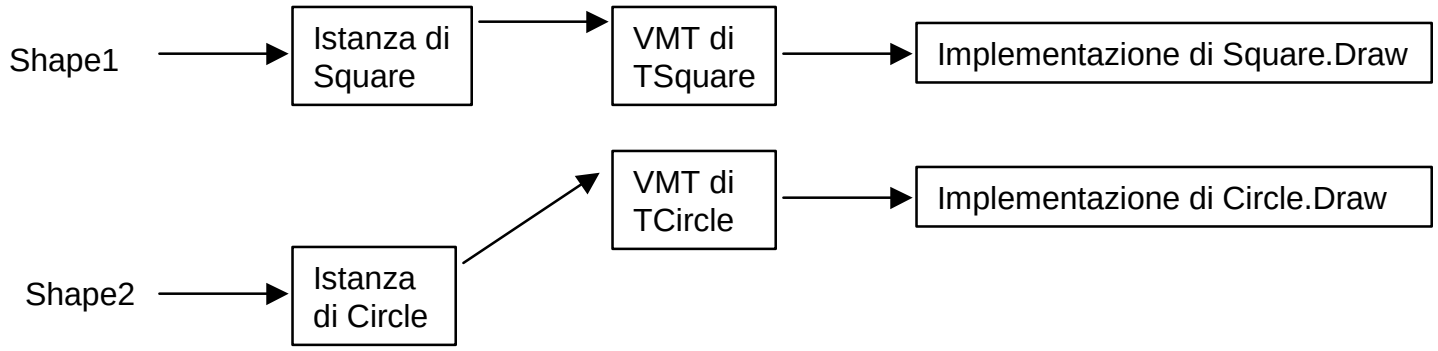
- **Quando il compilatore incontra una chiamata ad un metodo virtuale genera un codice di questo tipo:**
  - Ricava dall'istanza su cui il metodo viene invocato l'indirizzo della VMT della classe
  - Ricava dalla VMT l'indirizzo del metodo contenuto nello slot corrispondente al metodo invocato (usando la symbol table)
  - Chiama la subroutine all'indirizzo così ottenuto
- **Riassumendo: chiama il metodo referenziato dallo slot #n della VMT associata all'istanza**
- **Questa chiamata indiretta realizza il late binding: in pratica il metdo che viene effettivamente chiamato dipende dal contenuto della VMT e quidi dal contenuto dell'istanza che contiene il puntatore alla VMT**
- **Questa tecnica consente di realizzare il polimorfismo in linguaggi ad oggetti di tipo statico (cioè compilati): il comportamento polimorfo deriva dal fatto che ogni istanza contiene un informazione della classe a cui appartiene, sotto forma di VMT**
- **Classi derivate dalla stessa classe genitrice hanno la prima parte della VMT identica con i metodi nello stesso ordine**

Esempio di late binding

- **Consideriamo un caso tipico di polimorfismo: classe base TShape che definisce un metodo virtuale Draw e due classi derivate, TSquare e TCircle, che ridefiniscono Draw.**

```
var Shape1,Shape2:TShape;
...
Shape1:=TSquare.Create;
Shape2:=TCircle.Create;
```

- **Se invochiamo Shape1.Draw otterremo un quadrato mentre se invochiamo Shape2.Draw otteniamo un cerchio infatti:**

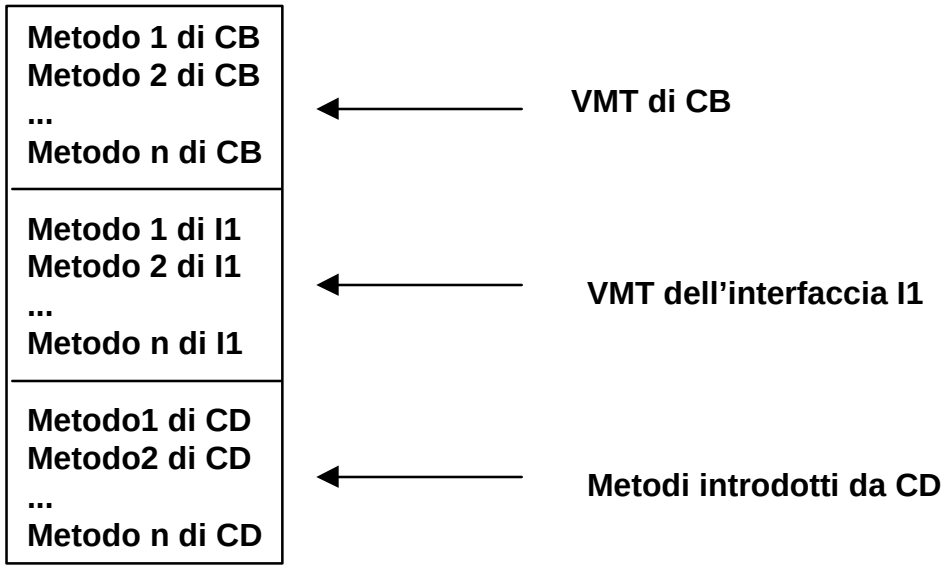


Classi virtuali pure e astratte

- **Una classe virtuale pura è una classe che non ha attributi e che comprende solo metodi virtuali**
- **A tutti gli effetti coincide con una VMT e un'istanza di tale classe è costituita solamente dal puntatore alla VMT**
- **Si parla di classe virtuale astratta se tutti i metodi di una classe virtuale pura sono definiti come abstract e quindi non hanno implementazione**
- **In pratica una classe virtuale astratta definisce la struttura di una VMT con i metodi messi in un ordine ben definito**
- **Le interfacce equivalgono a classi virtuali astratte**
- **Tutti gli oggetti che implementano una determinata interfaccia hanno una porzione di VMT identica, con i metodi definiti nell'interfaccia in posizioni ben precise.**

Struttura di una VMT tipica

- **Consideriamo una classe CD derivata dalla classe base CB, che implementa l'interfaccia I1 e definisce alcuni nuovi metodi:**



Oggetti e DLL

- Le DLL possono esportare solo funzioni e non strutture dati
- Questo pone un problema a chi fa uso di linguaggi ad oggetti: come si può accedere a oggetti situati all'interno di DLL?
- La tecnica più diffusa si basa sull'utilizzo di classi virtuali astratte
- Definiamo una classe virtuale astratta (CVA) con tutti i metodi che vogliamo rendere visibili all'esterno, compreso un distruttore
- Questa classe viene inserita in una unit che verrà inclusa sia nella DLL che nell EXE
- Nella DLL definiamo una classe concreta (CC) derivata da CVA
- Definiamo quindi una funzione "costruttore" che sarà l'unica funzione esportata dalla DLL e che restituisce un'istanza della classe CC
- Nell'EXE dichiariamo una variabile di tipo CVA gli assegniamo il risultato della funzione costruttore
- In base al polimorfismo possiamo ora invocare tutti i metodi di CC che derivano da CVA (in pratica la parte alta della VMT)
- Dal momento che tra i metodi di CVA c'è anche un distruttore possiamo eliminare l'oggetto quando non ci serve più