Alessio Bechini
- Corso di Fondamenti di Informatica II -

Il paradigma OO e le relative metodologie di progettazione

Programmazione orientata agli oggetti

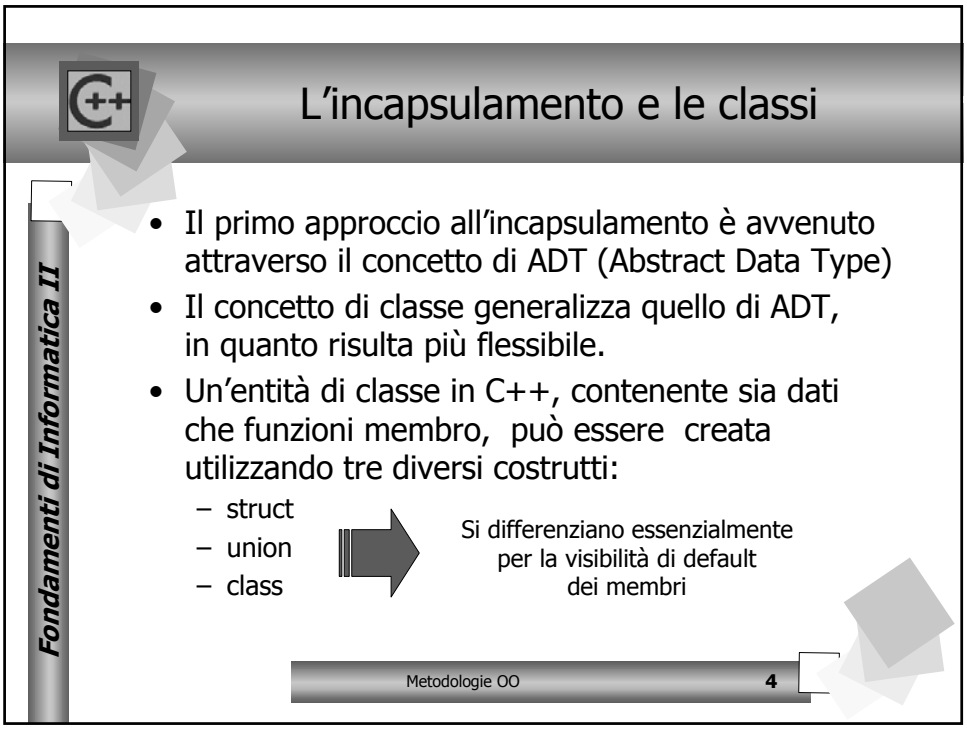
Fondamenti di Informatica II

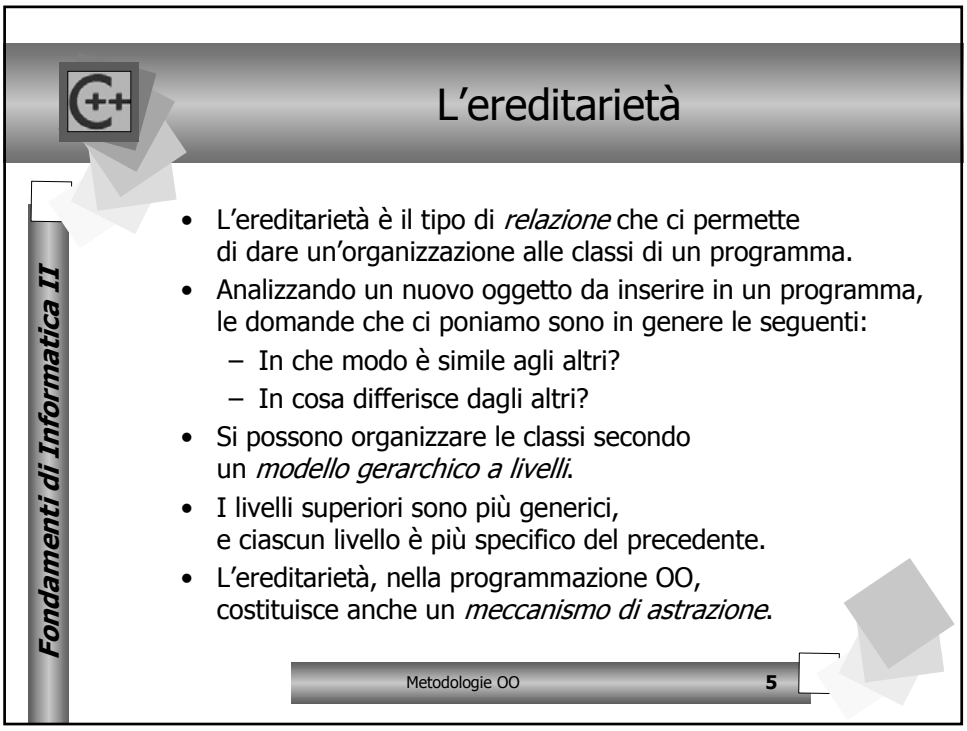
La programmazione ad oggetti (OOP)
è un paradigma di programmazione
che si ispira al modo in cui l'uomo crea modelli
per la comprensione del mondo reale

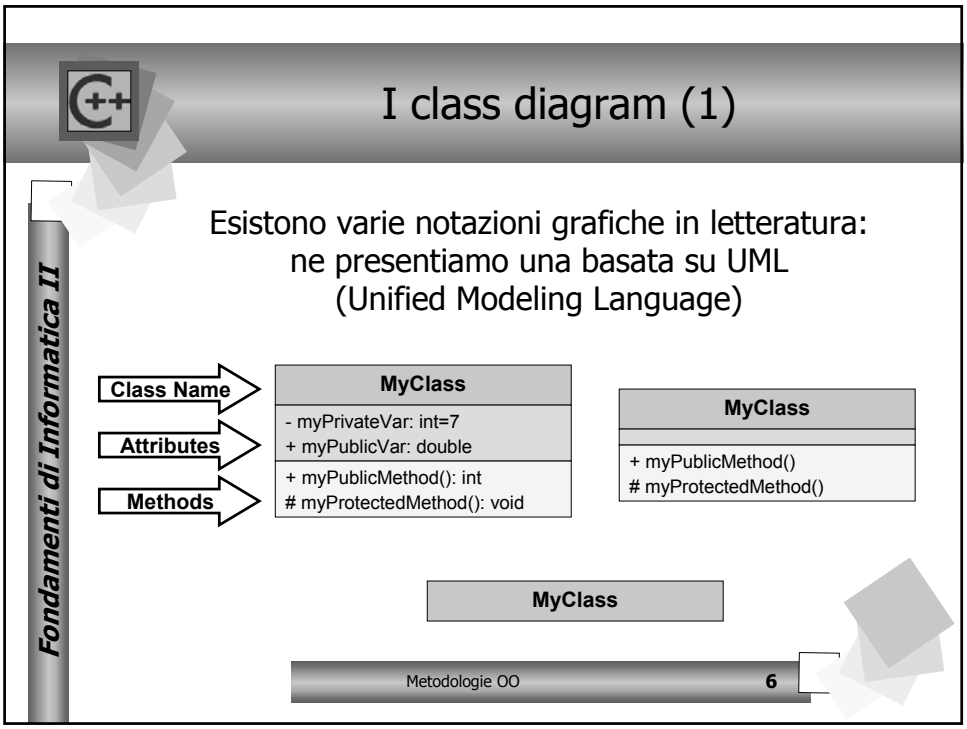
La OOP adotta un insieme di meccanismi
per gestire la complessità di un progetto software,
ricorrendo a regole per:

astrarre - generalizzare - classificare


Caratteristiche fondamentali
Fondamenti di Informatica II
• Incapsulamento: capacità di distinguere lo stato (e comportamento) *interno* di un oggetto dal suo stato (e comportamento) *esterno*
– Data hiding: capacità di nascondere i dettagli relativi allo stato interno di un oggetto
– Estendibilità dei tipi: capacità di aggiungere tipi definiti dall'utente ai tipi nativi del linguaggio
• Ereditarietà: capacità di creare nuovi tipi importando o riusando la descrizione di tipi già esistenti
• Polimorfismo: capacità di richiamare una *stessa funzionalità* (che può richiedere *implementazioni diverse* a seconda del *contesto* in cui è utilizzata) attraverso un unico identificatore. La scelta dell'implementazione appropriata può avvenire a tempo di compilazione o di esecuzione.
Metodologie OO
3


L'incapsulamento e le classi
Fondamenti di Informatica II
• Il primo approccio all'incapsulamento è avvenuto attraverso il concetto di ADT (Abstract Data Type)
• Il concetto di classe generalizza quello di ADT, in quanto risulta più flessibile.
• Un'entità di classe in C++, contenente sia dati che funzioni membro, può essere creata utilizzando tre diversi costrutti:
– struct
– union
– class
Si differenziano essenzialmente per la visibilità di default dei membri
Metodologie OO
4


L'ereditarietà
Fondamenti di Informatica II
• L'ereditarietà è il tipo di *relazione* che ci permette di dare un'organizzazione alle classi di un programma.
• Analizzando un nuovo oggetto da inserire in un programma, le domande che ci poniamo sono in genere le seguenti:
– In che modo è simile agli altri?
– In cosa differisce dagli altri?
• Si possono organizzare le classi secondo un *modello gerarchico a livelli*.
• I livelli superiori sono più generici, e ciascun livello è più specifico del precedente.
• L'ereditarietà, nella programmazione OO, costituisce anche un *meccanismo di astrazione*.
Metodologie OO
5


I class diagram (1)
Fondamenti di Informatica II
Esistono varie notazioni grafiche in letteratura:
ne presentiamo una basata su UML
(Unified Modeling Language)
Class Name
Attributes
Methods
MyClass
- myPrivateVar: int=7
+ myPublicVar: double
+ myPublicMethod(): int
# myProtectedMethod(): void
MyClass
+ myPublicMethod()
# myProtectedMethod()
MyClass
Metodologie OO
6

I class diagram (2)

Fondamenti di Informatica II

In UML, si possono individuare vari tipi di relazione tra classi:

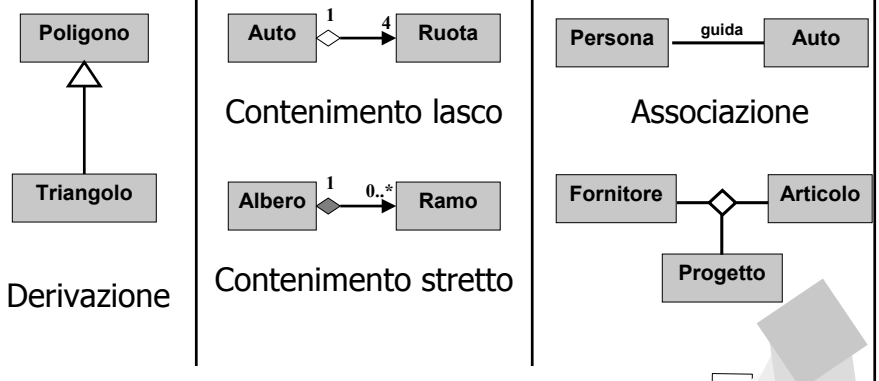
- Derivazione (Gen.-Spec.): indica se una classe è sottoclasse diretta (figlio) di una classe base (padre).
- Composizione (aggregazione, uso): indica se una classe tra i suoi campi possiede (o riferisce) un'istanza (o più istanze) di un'altra classe.
  - esclusivo (*aggregation*, part-of, contenimento stretto)
  - eventualmente condiviso (*acquaintance*, conoscenza, contenimento lasco)
- Associazione: legame *semantico* tra classi, caratterizzato da un nome, dai ruoli giocati dalle classi, dalla molteplicità, dalla direzionalità.

Metodologie OO 7

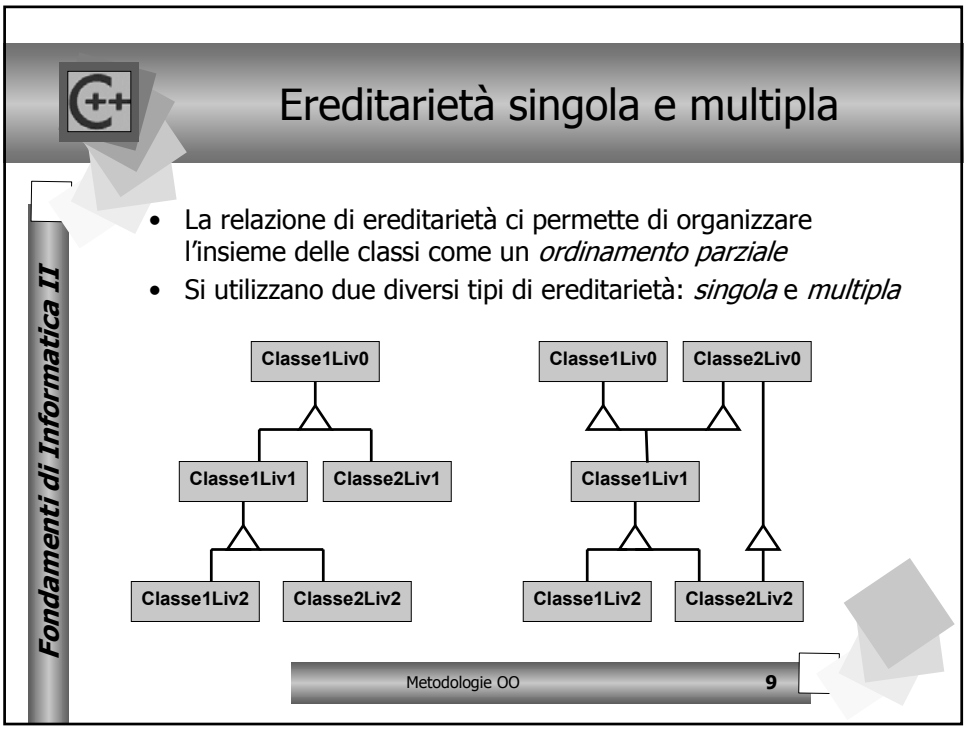
I class diagram (3)

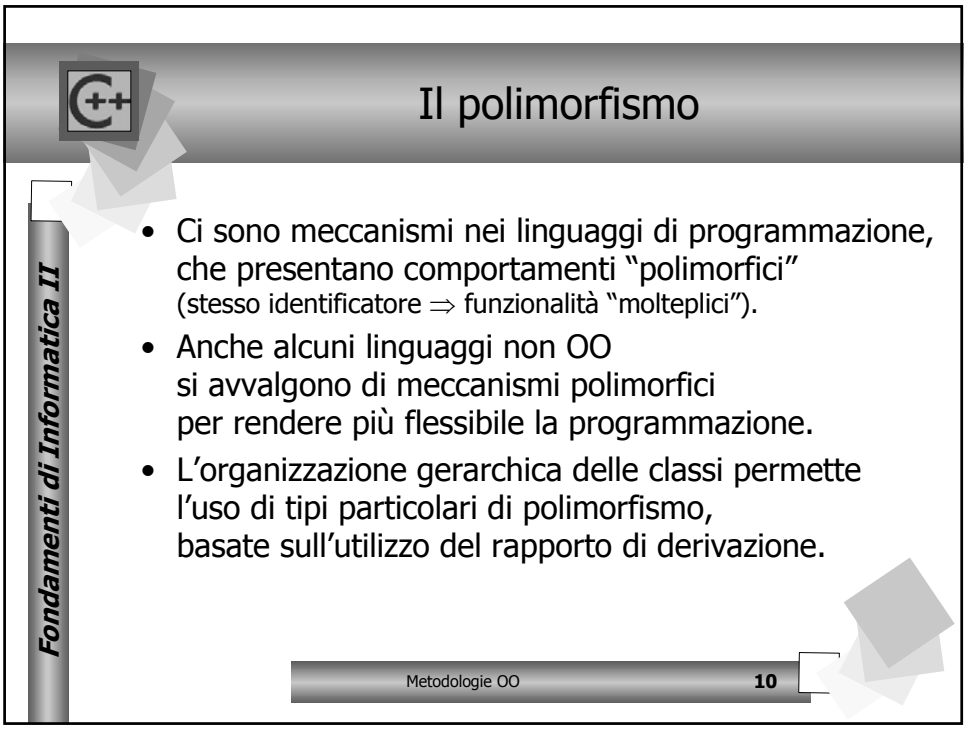
Fondamenti di Informatica II

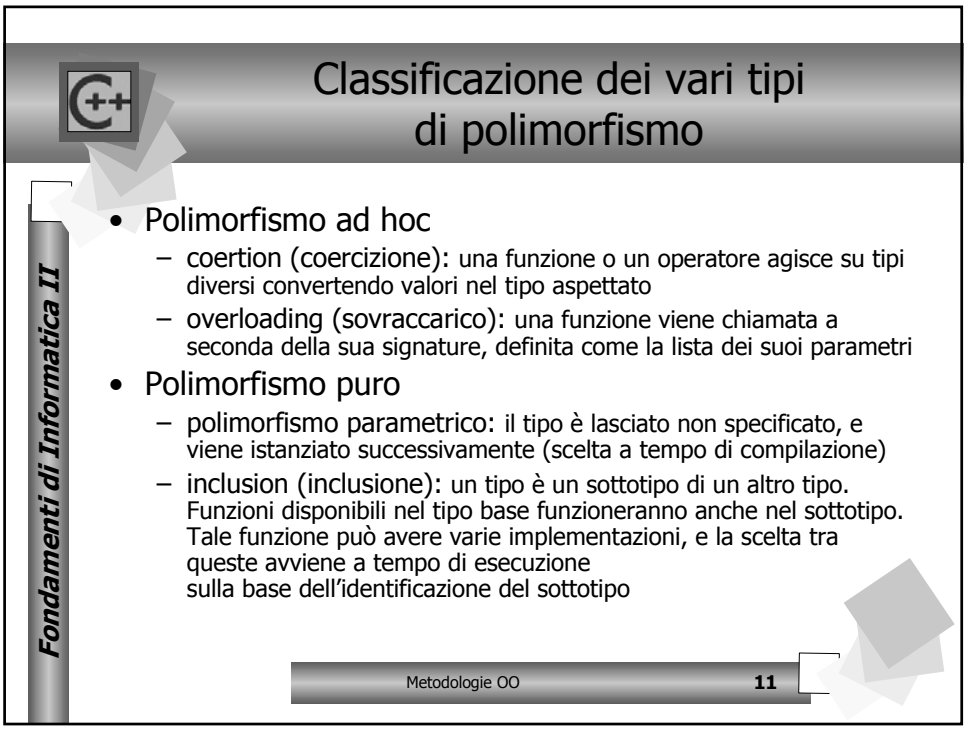
Vari tipi di relazione tra classi in UML

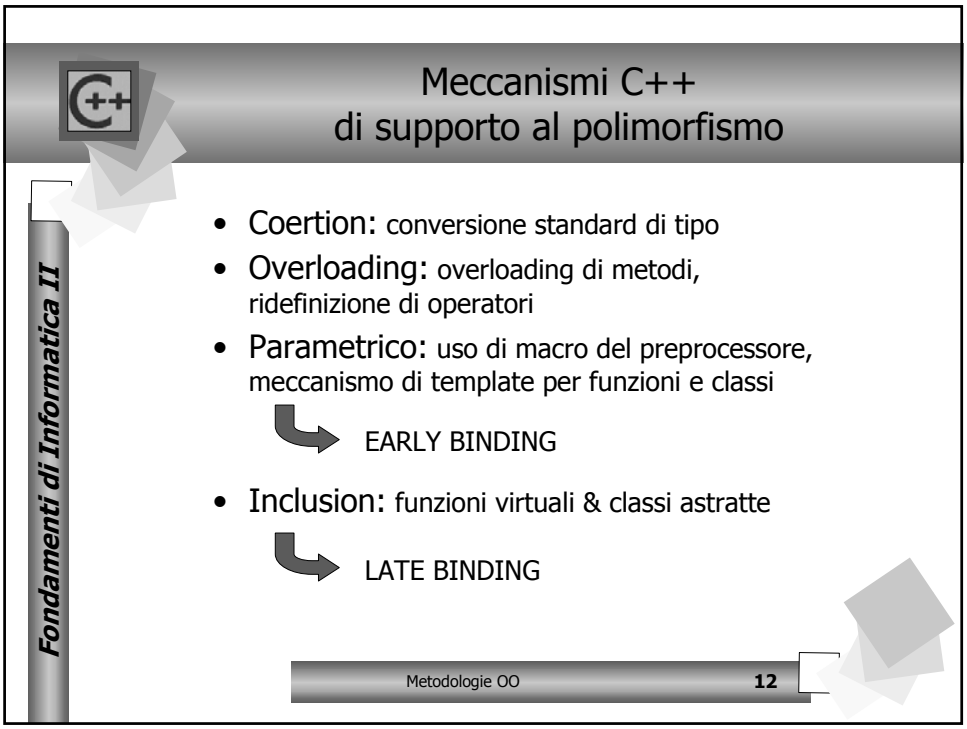


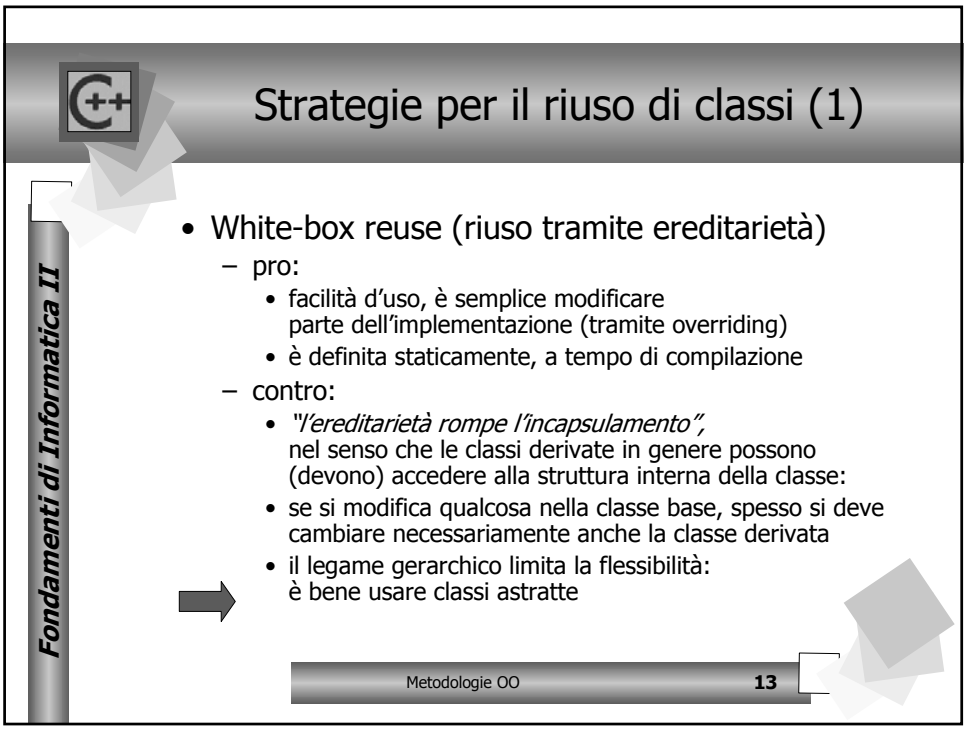
Metodologie OO 8

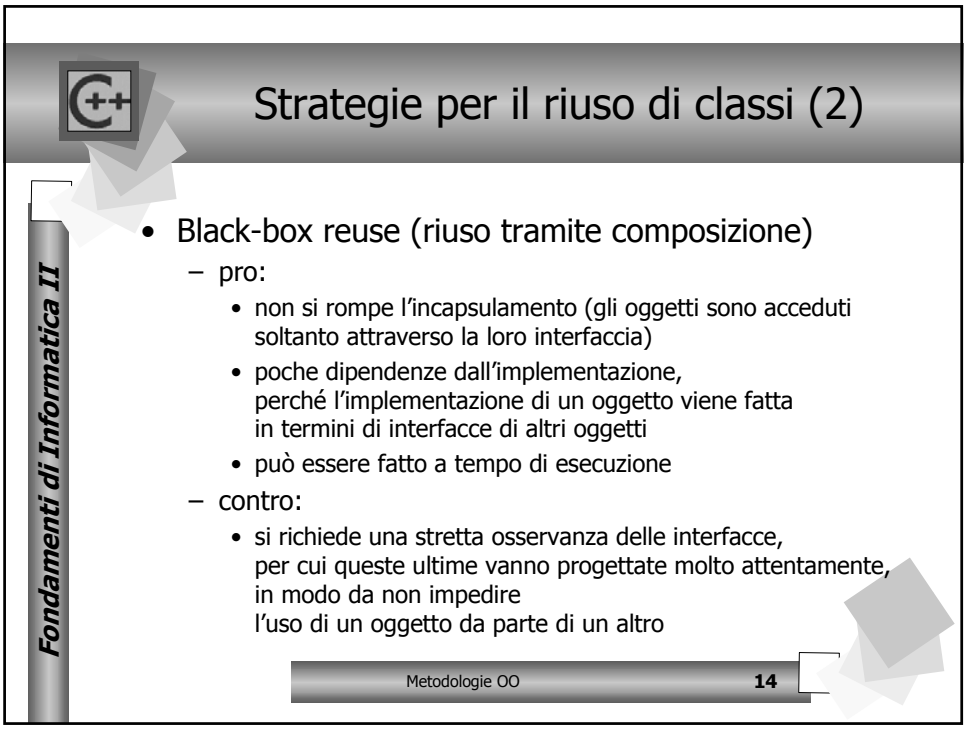

Ereditarietà singola e multipla
Fondamenti di Informatica II
La relazione di ereditarietà ci permette di organizzare l'insieme delle classi come un *ordinamento parziale*
Si utilizzano due diversi tipi di ereditarietà: *singola* e *multipla*
Classe1Liv0
Classe1Liv1
Classe2Liv1
Classe1Liv2
Classe2Liv2
Classe1Liv0
Classe2Liv0
Classe1Liv1
Classe1Liv2
Classe2Liv2
Metodologie OO
9

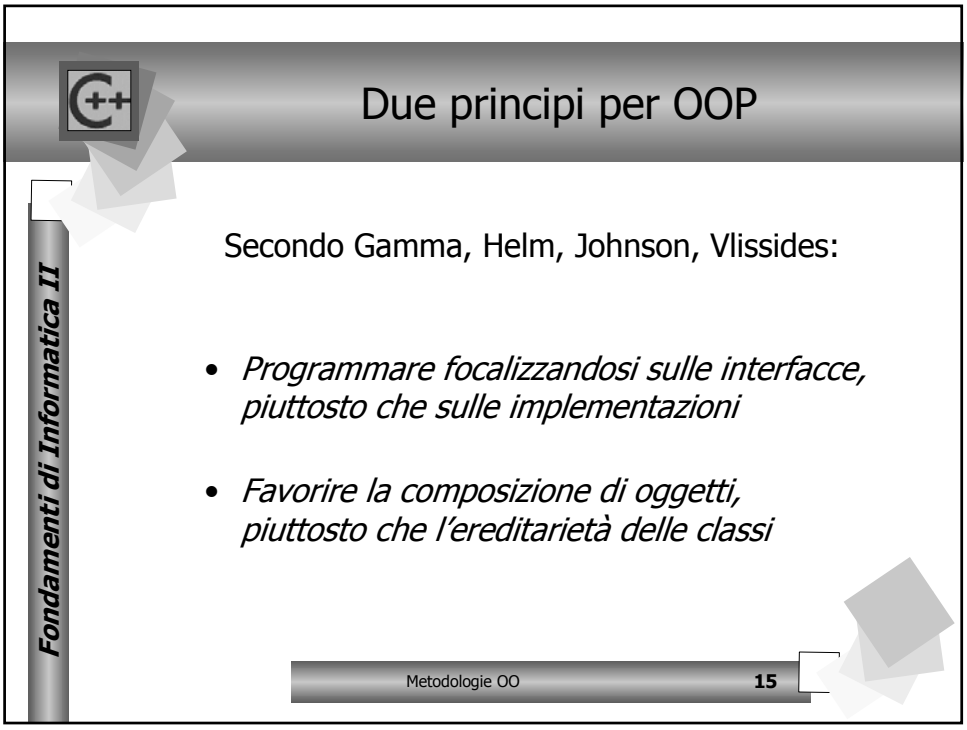

Il polimorfismo
Fondamenti di Informatica II
Ci sono meccanismi nei linguaggi di programmazione, che presentano comportamenti "polimorfici"
(stesso identificatore ⇒ funzionalità "molteplici").
Anche alcuni linguaggi non OO si avvalgono di meccanismi polimorfici per rendere più flessibile la programmazione.
L'organizzazione gerarchica delle classi permette l'uso di tipi particolari di polimorfismo, basate sull'utilizzo del rapporto di derivazione.
Metodologie OO
10

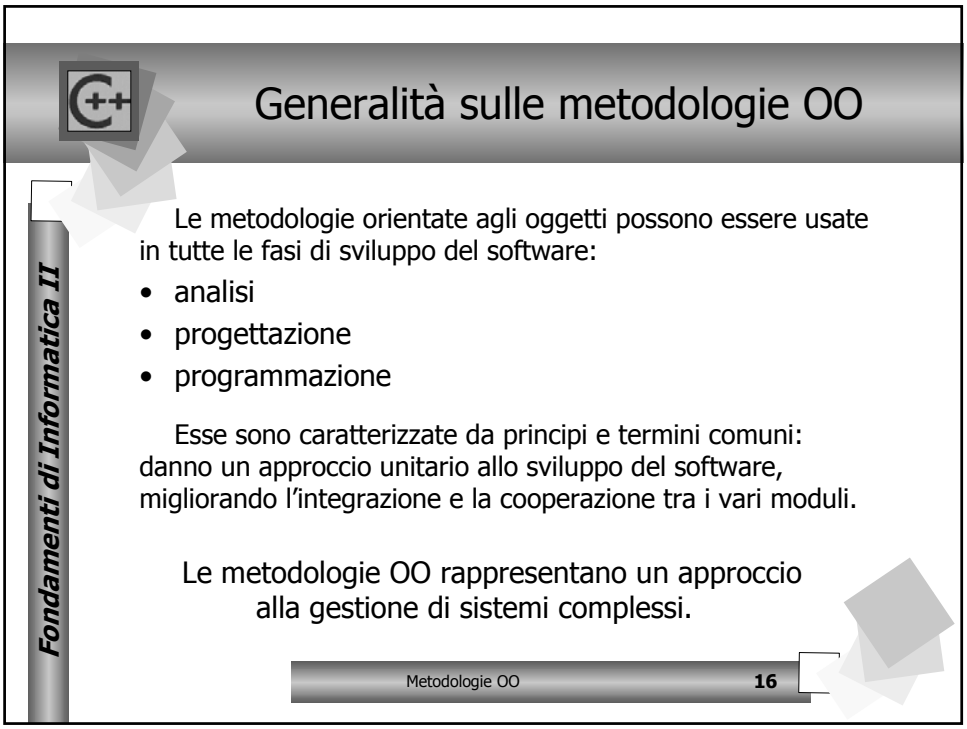

Classificazione dei vari tipi di polimorfismo
Fondamenti di Informatica II
• Polimorfismo ad hoc
– coertion (coercizione): una funzione o un operatore agisce su tipi diversi convertendo valori nel tipo aspettato
– overloading (sovraccarico): una funzione viene chiamata a seconda della sua signature, definita come la lista dei suoi parametri
• Polimorfismo puro
– polimorfismo parametrico: il tipo è lasciato non specificato, e viene istanziato successivamente (scelta a tempo di compilazione)
– inclusion (inclusione): un tipo è un sottotipo di un altro tipo. Funzioni disponibili nel tipo base funzioneranno anche nel sottotipo. Tale funzione può avere varie implementazioni, e la scelta tra queste avviene a tempo di esecuzione sulla base dell'identificazione del sottotipo
Metodologie OO 11

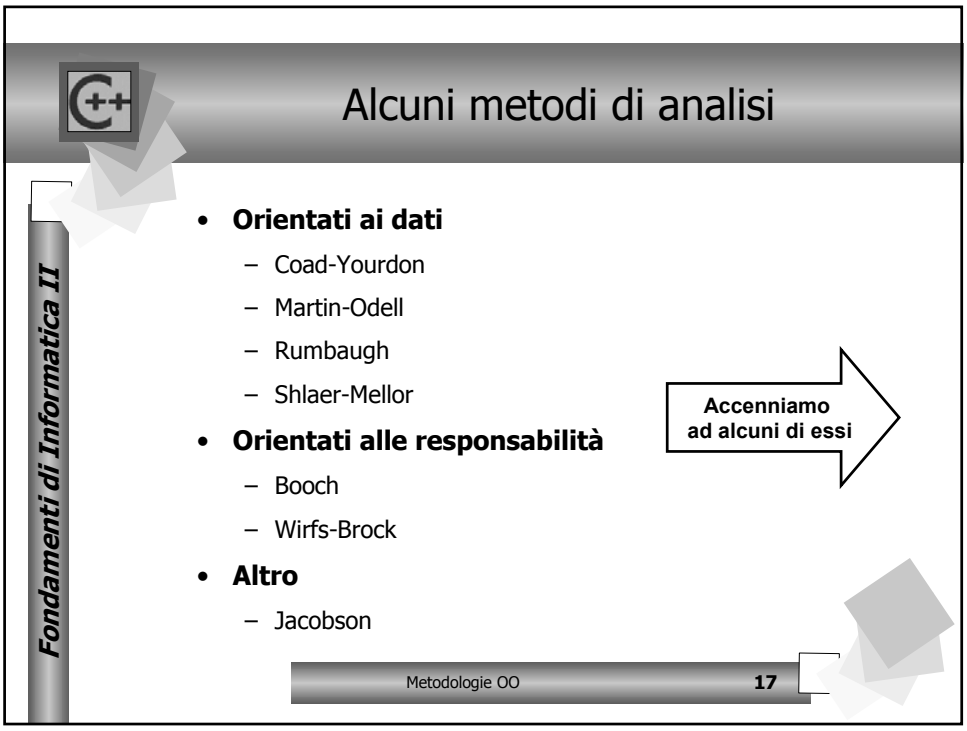

Meccanismi C++ di supporto al polimorfismo
Fondamenti di Informatica II
• Coertion: conversione standard di tipo
• Overloading: overloading di metodi, ridefinizione di operatori
• Parametrico: uso di macro del preprocessore, meccanismo di template per funzioni e classi
EARLY BINDING
• Inclusion: funzioni virtuali & classi astratte
LATE BINDING
Metodologie OO 12


Strategie per il riuso di classi (1)
Fondamenti di Informatica II
• White-box reuse (riuso tramite ereditarietà)
– pro:
• facilità d'uso, è semplice modificare parte dell'implementazione (tramite overriding)
• è definita staticamente, a tempo di compilazione
– contro:
• "l'ereditarietà rompe l'incapsulamento", nel senso che le classi derivate in genere possono (devono) accedere alla struttura interna della classe:
• se si modifica qualcosa nella classe base, spesso si deve cambiare necessariamente anche la classe derivata
• il legame gerarchico limita la flessibilità: è bene usare classi astratte
Metodologie OO
13

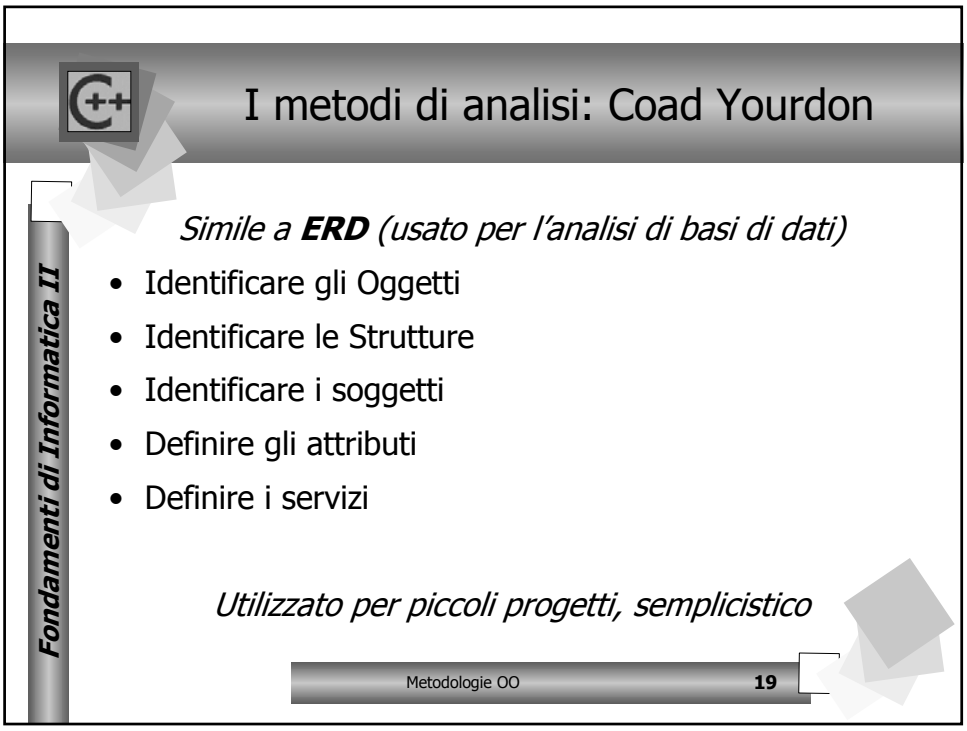

Strategie per il riuso di classi (2)
Fondamenti di Informatica II
• Black-box reuse (riuso tramite composizione)
– pro:
• non si rompe l'incapsulamento (gli oggetti sono acceduti soltanto attraverso la loro interfaccia)
• poche dipendenze dall'implementazione, perché l'implementazione di un oggetto viene fatta in termini di interfacce di altri oggetti
• può essere fatto a tempo di esecuzione
– contro:
• si richiede una stretta osservanza delle interfacce, per cui queste ultime vanno progettate molto attentamente, in modo da non impedire l'uso di un oggetto da parte di un altro
Metodologie OO
14

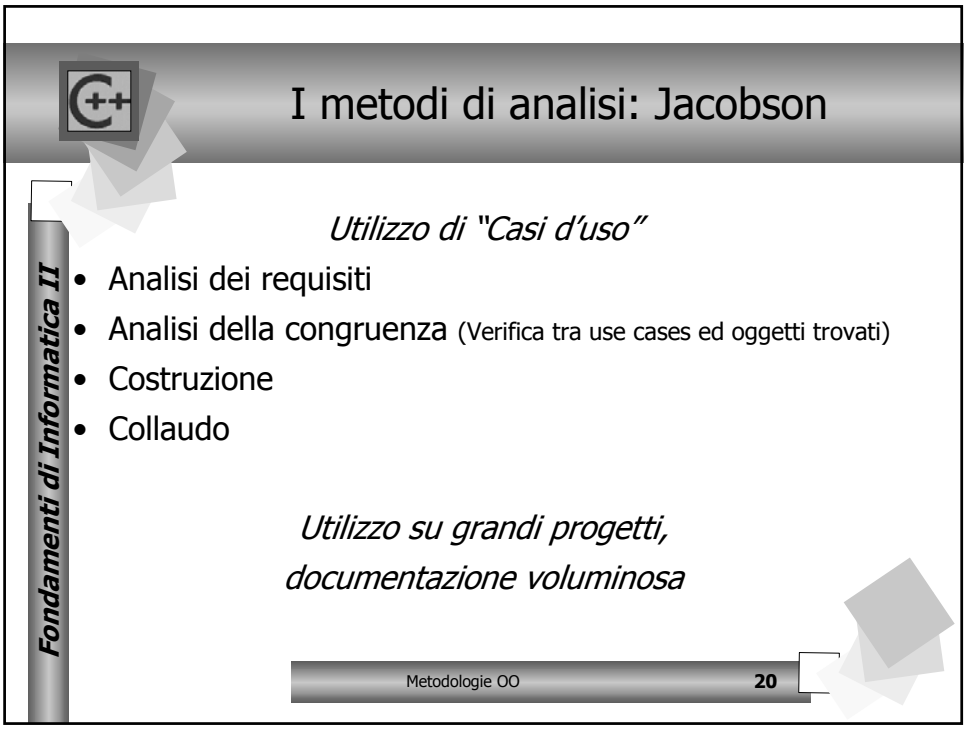

Due principi per OOP
Fondamenti di Informatica II
Secondo Gamma, Helm, Johnson, Vlissides:
• Programmare focalizzandosi sulle interfacce, piuttosto che sulle implementazioni
• Favorire la composizione di oggetti, piuttosto che l'ereditarietà delle classi
Metodologie OO
15


Generalità sulle metodologie OO
Fondamenti di Informatica II
Le metodologie orientate agli oggetti possono essere usate in tutte le fasi di sviluppo del software:
• analisi
• progettazione
• programmazione
Esse sono caratterizzate da principi e termini comuni: danno un approccio unitario allo sviluppo del software, migliorando l'integrazione e la cooperazione tra i vari moduli.
Le metodologie OO rappresentano un approccio alla gestione di sistemi complessi.
Metodologie OO
16

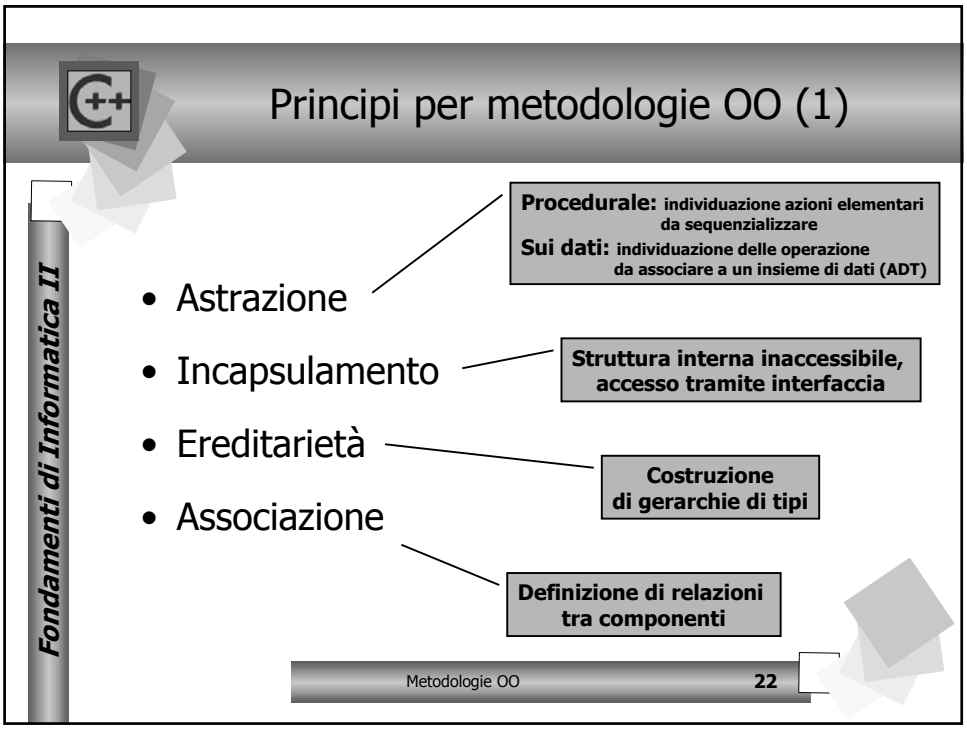

Alcuni metodi di analisi
Fondamenti di Informatica II
Orientati ai dati
Coad-Yourdon
Martin-Odell
Rumbaugh
Shlaer-Mellor
Orientati alle responsabilità
Booch
Wirfs-Brock
Altro
Jacobson
Accenniamo ad alcuni di essi
Metodologie OO
17


I metodi di analisi: Booch
Fondamenti di Informatica II
Identificare Classi ed Oggetti (Dominio del problema)
Identificare la responsabilità delle Classi (Descrivere ciclo di vita degli oggetti)
Identificare le relazioni tra classi ed oggetti
Realizzare Classi ed Oggetti
Utilizzato su piccoli progetti in C++
Metodologie OO
18


I metodi di analisi: Coad Yourdon
Simile a ERD (usato per l'analisi di basi di dati)
• Identificare gli Oggetti
• Identificare le Strutture
• Identificare i soggetti
• Definire gli attributi
• Definire i servizi
Utilizzato per piccoli progetti, semplicistico
Fondamenti di Informatica II
Metodologie OO 19


I metodi di analisi: Jacobson
Utilizzo di "Casi d'uso"
• Analisi dei requisiti
• Analisi della congruenza (Verifica tra use cases ed oggetti trovati)
• Costruzione
• Collaudo
Utilizzo su grandi progetti,
documentazione voluminosa
Fondamenti di Informatica II
Metodologie OO 20


I metodi di analisi: Rumbaugh
Fondamenti di Informatica II
Basato su tre modelli
• Modello degli oggetti
• Modello Dinamico
• Modello Funzionale
Prevede le seguenti azioni:
• Analisi (Definizione del sistema -Definizione oggetti, relazioni e flussi di controllo)
• Progettazione del sistema (Definizione sottosistemi)
• Progettazione degli oggetti
Utilizzato su grandi progetti, con approccio rigoroso
Metodologie OO
21


Principi per metodologie OO (1)
Fondamenti di Informatica II
• Astrazione
• Incapsulamento
• Ereditarietà
• Associazione
Procedurale: individuazione azioni elementari da sequenzializzare
Sui dati: individuazione delle operazione da associare a un insieme di dati (ADT)
Struttura interna inaccessibile, accesso tramite interfaccia
Costruzione di gerarchie di tipi
Definizione di relazioni tra componenti
Metodologie OO
22

Principi per metodologie OO (2)

Fondamenti di Informatica II

- Comunicazione con messaggi — **Interazione attraverso invocazione di metodi pubblici**
- Metodi di organizzazione — **Strutturazione della conoscenza**
- Riduzione in scala — **Suddivisione in sottodomini e sottoproblemi**
- Categorie di comportamento — **Evoluzione temporale, relazioni di causalità, similitudine di funzionalità**

Metodologie OO 23

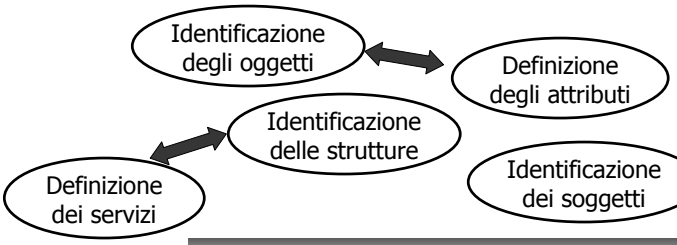
Analisi orientata agli oggetti

Fondamenti di Informatica II

Con il termine "analisi" si intende
lo studio del dominio di un sistema.

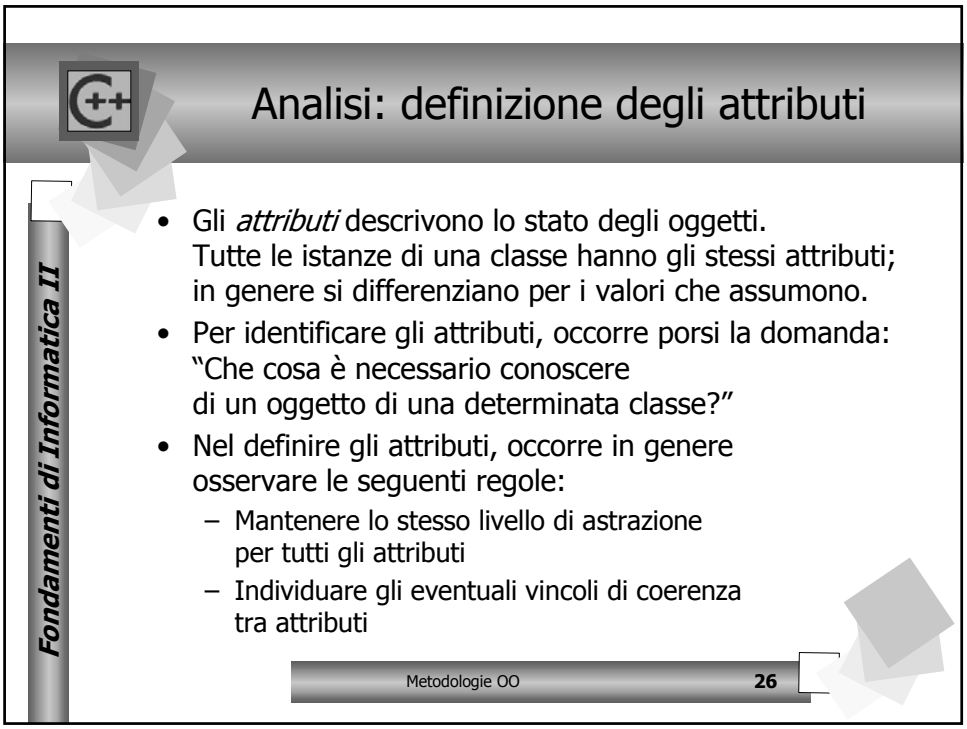
La fase di analisi deve portare a una descrizione completa e coerente del sistema e di ciò che esso deve fare: tale descrizione deve essere indipendente dal linguaggio di programmazione usato.

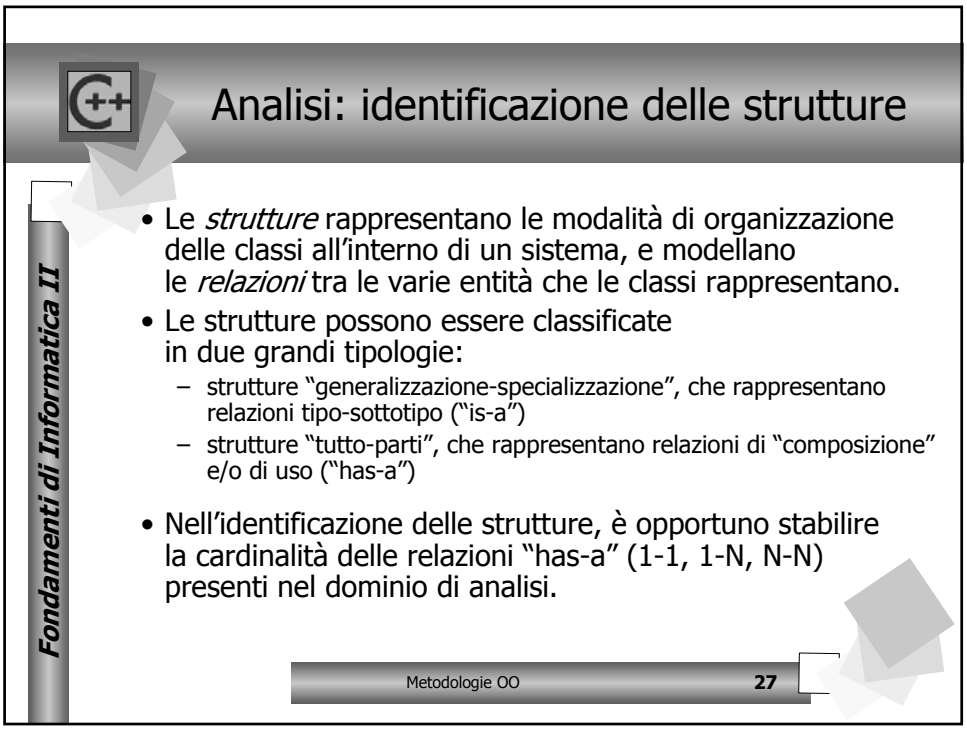
I passi da utilizzare sono i seguenti:

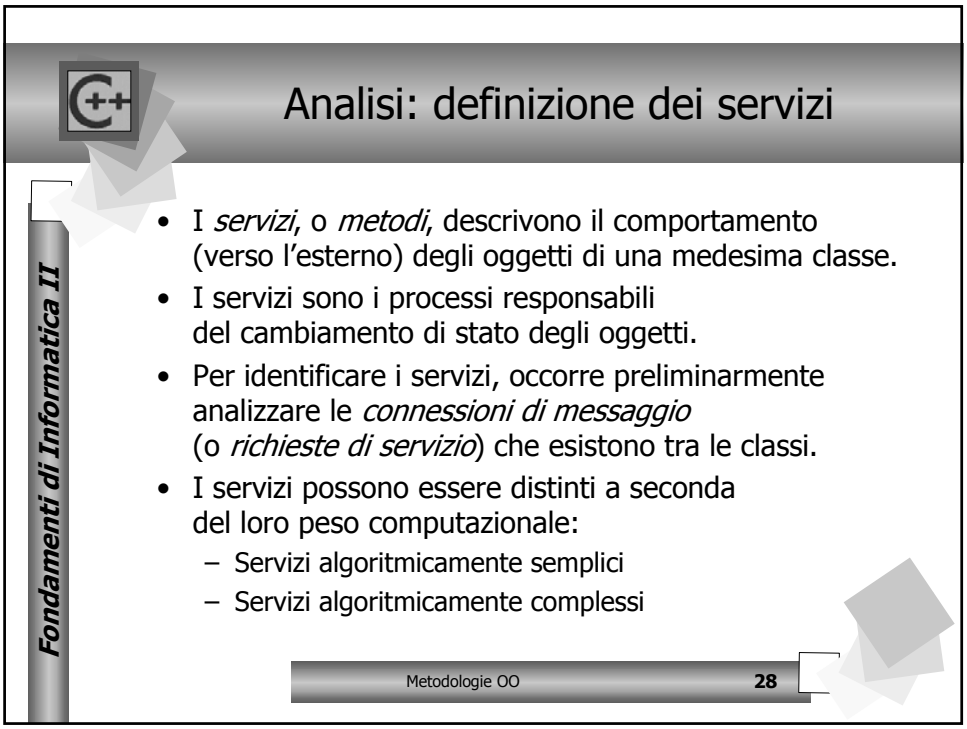


Metodologie OO 24

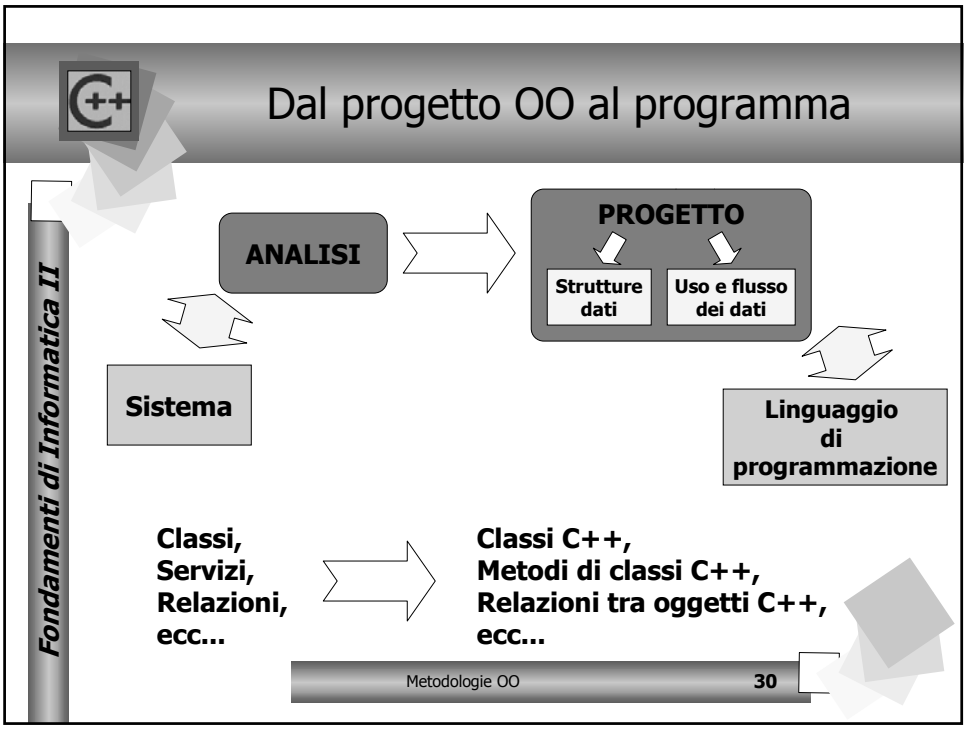

Analisi: identificazione degli oggetti
Fondamenti di Informatica II
• Gli *oggetti* rappresentano i componenti del sistema, e possono essere entità reali o astratte.
• Per semplificare la loro gestione, vengono raggruppati in *classi*.
• Non ci sono regole universali per l'identificazione delle classi, ma si possono dare alcuni consigli:
– Studiare preventivamente il dominio di applicazione
– Dialogare con gli esperti del dominio
– Cercare sistemi con qualche analogia con quello studiato
– Porsi la domanda: "Di quale entità è rilevante mantenere l'informazione?"
Metodologie OO 25

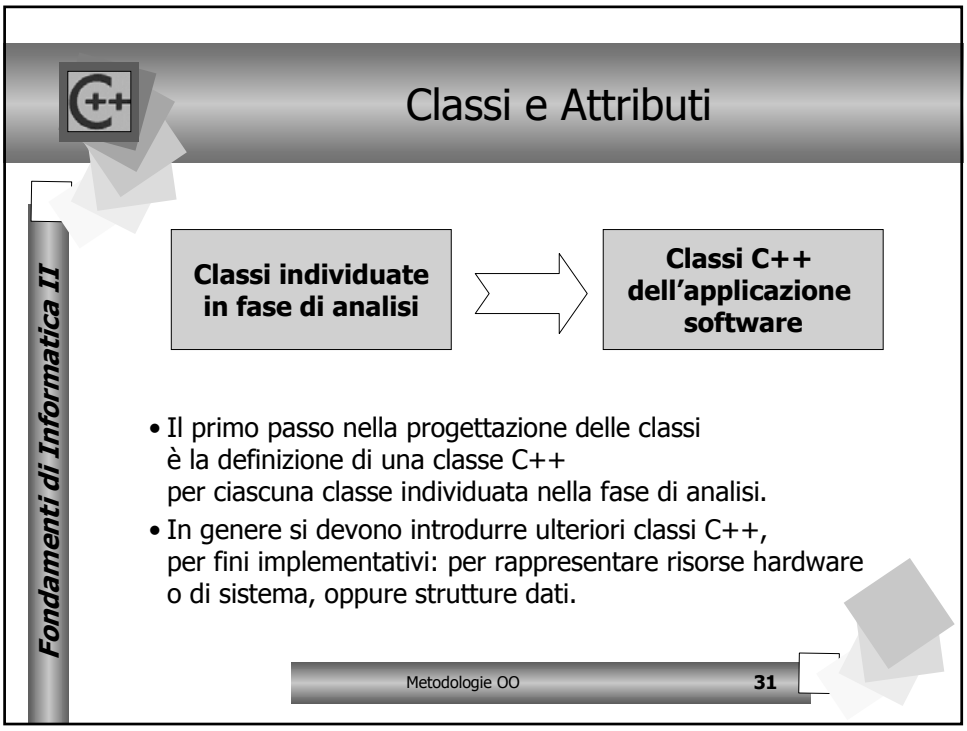

Analisi: definizione degli attributi
Fondamenti di Informatica II
• Gli *attributi* descrivono lo stato degli oggetti. Tutte le istanze di una classe hanno gli stessi attributi; in genere si differenziano per i valori che assumono.
• Per identificare gli attributi, occorre porsi la domanda: "Che cosa è necessario conoscere di un oggetto di una determinata classe?"
• Nel definire gli attributi, occorre in genere osservare le seguenti regole:
– Mantenere lo stesso livello di astrazione per tutti gli attributi
– Individuare gli eventuali vincoli di coerenza tra attributi
Metodologie OO 26

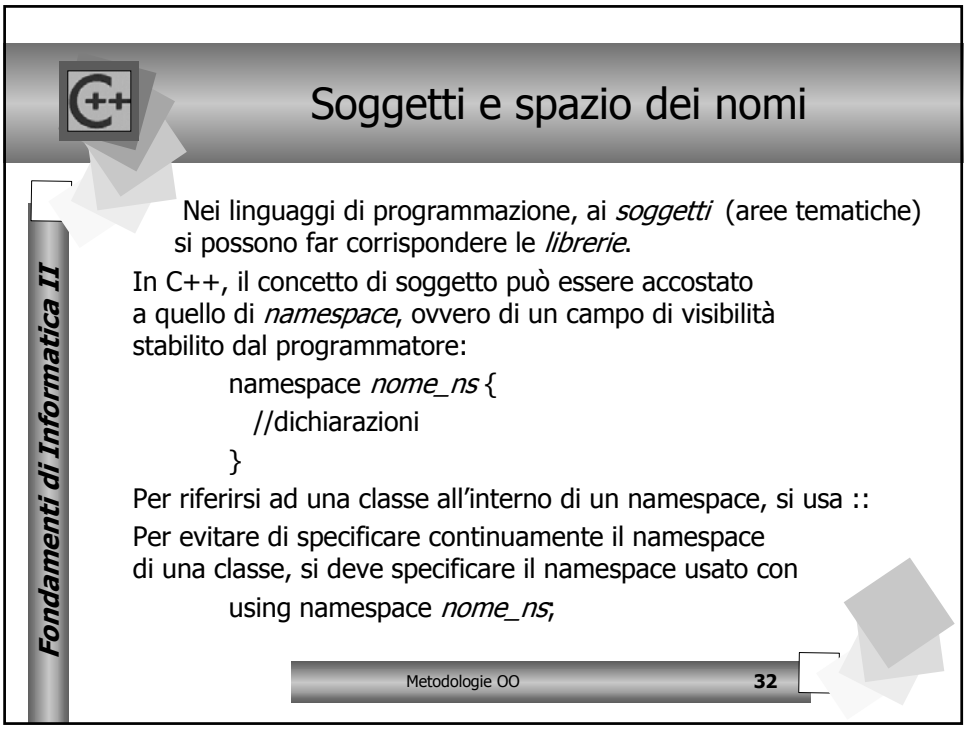

Analisi: identificazione delle strutture
Fondamenti di Informatica II
• Le *strutture* rappresentano le modalità di organizzazione delle classi all'interno di un sistema, e modellano le *relazioni* tra le varie entità che le classi rappresentano.
• Le strutture possono essere classificate in due grandi tipologie:
– strutture "generalizzazione-specializzazione", che rappresentano relazioni tipo-sottotipo ("is-a")
– strutture "tutto-parti", che rappresentano relazioni di "composizione" e/o di uso ("has-a")
• Nell'identificazione delle strutture, è opportuno stabilire la cardinalità delle relazioni "has-a" (1-1, 1-N, N-N) presenti nel dominio di analisi.
Metodologie OO
27

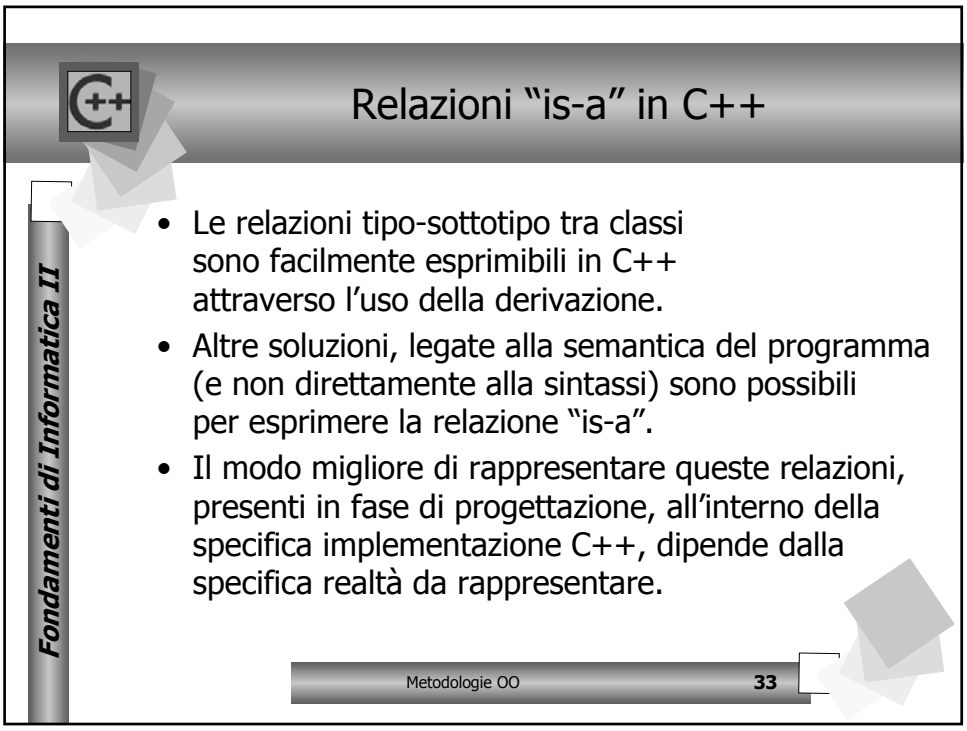

Analisi: definizione dei servizi
Fondamenti di Informatica II
• I *servizi*, o *metodi*, descrivono il comportamento (verso l'esterno) degli oggetti di una medesima classe.
• I servizi sono i processi responsabili del cambiamento di stato degli oggetti.
• Per identificare i servizi, occorre preliminarmente analizzare le *connessioni di messaggio* (o *richieste di servizio*) che esistono tra le classi.
• I servizi possono essere distinti a seconda del loro peso computazionale:
– Servizi algoritmicamente semplici
– Servizi algoritmicamente complessi
Metodologie OO
28

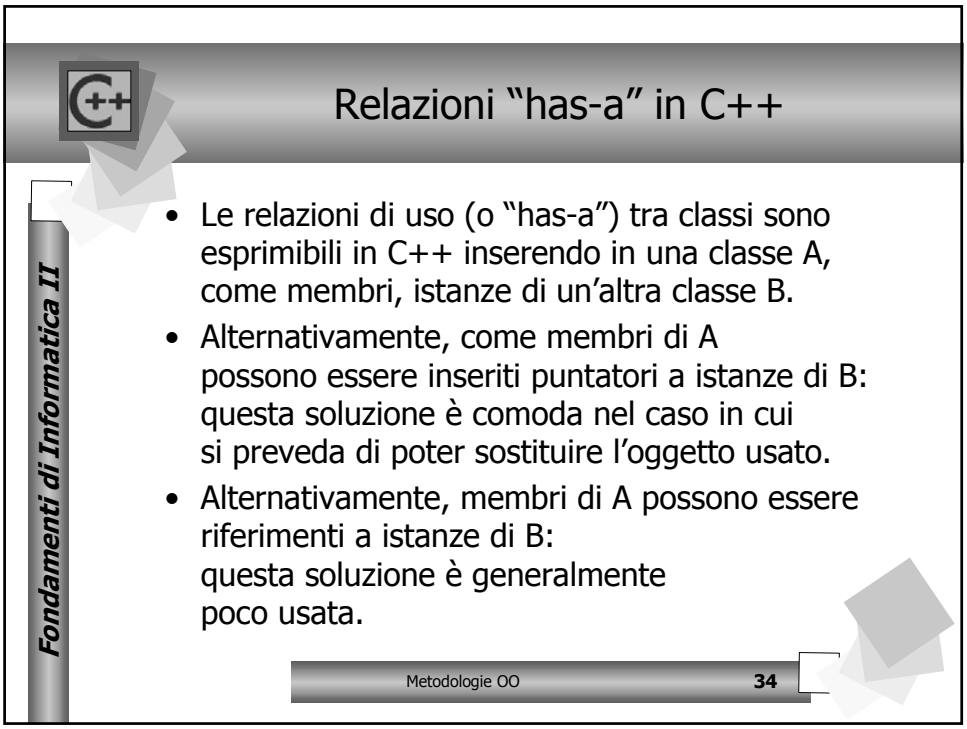

Analisi: identificazione dei soggetti
Fondamenti di Informatica II
• I *soggetti* corrispondono alle "aree tematiche" all'interno di un sistema, e hanno lo scopo di permettere la suddivizione in sotto-sistemi più facilmente maneggiabili.
• Un soggetto contiene più classi, e una stessa classe può appartenere a più soggetti.
• Quando il numero delle classi aumenta eccessivamente, i soggetti aiutano a controllare la complessità del sistema
• Sistemi semplici non richiedono la definizione di soggetti.
• I soggetti devono rispondere ai seguenti requisiti:
– appartenenza al dominio dell'applicazione
– minimizzazione delle inter-dipendenze con altri soggetti
Metodologie OO 29

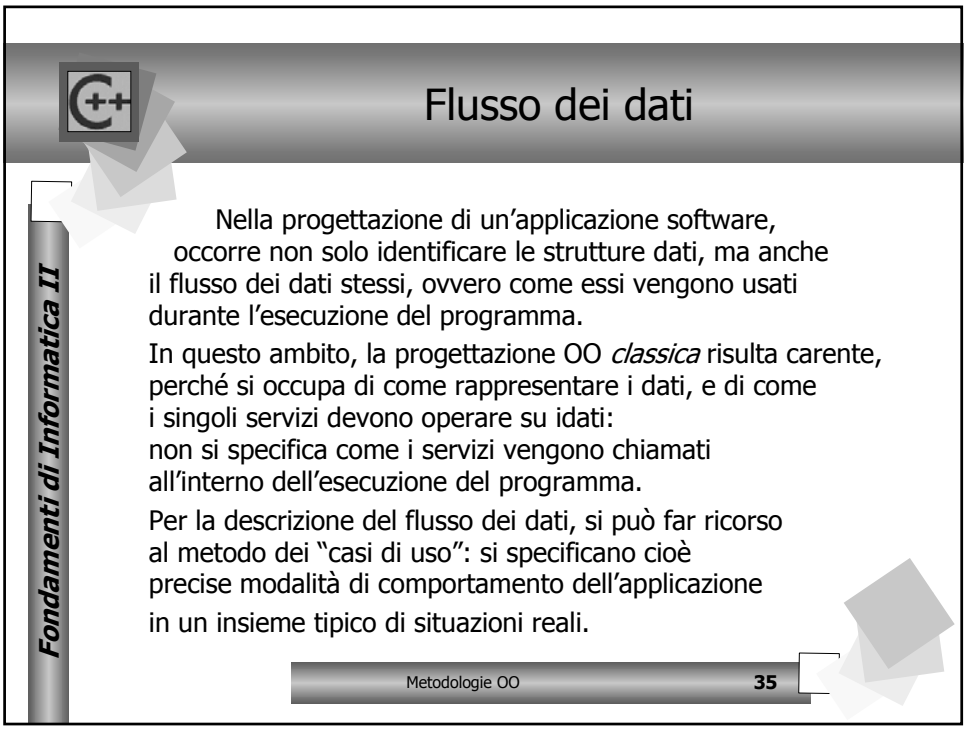

Dal progetto OO al programma
Fondamenti di Informatica II
ANALISI
PROGETTO
Strutture dati
Uso e flusso dei dati
Sistema
Linguaggio di programmazione
Classi, Servizi, Relazioni, ecc...
Classi C++, Metodi di classi C++, Relazioni tra oggetti C++, ecc...
Metodologie OO 30


Classi e Attributi
Fondamenti di Informatica II
Classi individuate in fase di analisi
Classi C++ dell'applicazione software
• Il primo passo nella progettazione delle classi è la definizione di una classe C++ per ciascuna classe individuata nella fase di analisi.
• In genere si devono introdurre ulteriori classi C++, per fini implementativi: per rappresentare risorse hardware o di sistema, oppure strutture dati.
Metodologie OO 31


Soggetti e spazio dei nomi
Fondamenti di Informatica II
Nei linguaggi di programmazione, ai *soggetti* (aree tematiche) si possono far corrispondere le *librerie*.
In C++, il concetto di soggetto può essere accostato a quello di *namespace*, ovvero di un campo di visibilità stabilito dal programmatore:
namespace *nome_ns* {
//dichiarazioni
}
Per riferirsi ad una classe all'interno di un namespace, si usa ::
Per evitare di specificare continuamente il namespace di una classe, si deve specificare il namespace usato con
using namespace *nome_ns*;
Metodologie OO 32


Relazioni "is-a" in C++
Fondamenti di Informatica II
• Le relazioni tipo-sottotipo tra classi sono facilmente esprimibili in C++ attraverso l'uso della derivazione.
• Altre soluzioni, legate alla semantica del programma (e non direttamente alla sintassi) sono possibili per esprimere la relazione "is-a".
• Il modo migliore di rappresentare queste relazioni, presenti in fase di progettazione, all'interno della specifica implementazione C++, dipende dalla specifica realtà da rappresentare.
Metodologie OO 33


Relazioni "has-a" in C++
Fondamenti di Informatica II
• Le relazioni di uso (o "has-a") tra classi sono esprimibili in C++ inserendo in una classe A, come membri, istanze di un'altra classe B.
• Alternativamente, come membri di A possono essere inseriti puntatori a istanze di B: questa soluzione è comoda nel caso in cui si preveda di poter sostituire l'oggetto usato.
• Alternativamente, membri di A possono essere riferimenti a istanze di B: questa soluzione è generalmente poco usata.
Metodologie OO 34


Flusso dei dati
Fondamenti di Informatica II
Nella progettazione di un'applicazione software, occorre non solo identificare le strutture dati, ma anche il flusso dei dati stessi, ovvero come essi vengono usati durante l'esecuzione del programma.
In questo ambito, la progettazione OO *classica* risulta carente, perché si occupa di come rappresentare i dati, e di come i singoli servizi devono operare su idati: non si specifica come i servizi vengono chiamati all'interno dell'esecuzione del programma.
Per la descrizione del flusso dei dati, si può far ricorso al metodo dei "casi di uso": si specificano cioè precise modalità di comportamento dell'applicazione in un insieme tipico di situazioni reali.
Metodologie OO
35