Linguaggi di Programmazione I – Lezione 4

Prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

18 marzo 2008

Panoramica della lezione

Modello orientato ad oggetti

Peculiarità della programmazione OO

Altri concetti OO

Bibliografia

Modello orientato ad oggetti

Introduzione

- *Scrivere un programma in un linguaggio OO (Object Oriented) non significa scrivere un programma OO.*
- *Scrivere un programma OO richiede prima un processo di OOAD (Object Oriented Analysis and Design).*

Oggetto

- È la rappresentazione di ogni cosa in un programma OO.
- Può essere il modello di un sensore, una finestra in una interfaccia utente, una struttura dati, ..., virtualmente tutto.
- Può essere visto come una scatola nera con pulsanti e spie.
- Per usare un oggetto bisogna sapere solo a cosa servono i pulsanti (quale premere per ottenere ciò che si vuole) ed il significato dei segnali luminosi delle spie.
- Ciò che importa è che l'oggetto esegua correttamente le proprie funzioni e si faccia carico delle proprie responsabilità.
- L'interno della scatola è completamente nascosto all'ambiente esterno.

In essenza:
Progettare un sistema OO consiste nell'identificare quali oggetti il sistema debba contenere, il comportamento e le responsabilità di ognuno di essi e come essi interagiscono tra loro.

Vantaggi della progettazione OO

1. Progettazione elegante e facile da capire.
2. Programmi eleganti e facili da capire.
3. Oggetti singoli realizzati e manutenuti in modo indipendente.
4. Librerie di oggetti facilmente riutilizzabili e riadattabili ad un nuovo progetto.
5. Facilità di modifica e di debug.

Ma attenzione:

OOAD non è la panacea di tutti i problemi associati alla produzione di software.

Linguaggi OO

- Smalltalk
- Eiffel
- C++
- Objective C
- Objective Pascal
- Java
- Ada
- ...

Il mercato ha decretato come vincenti *C++* e *Java*.

C++ vs Java

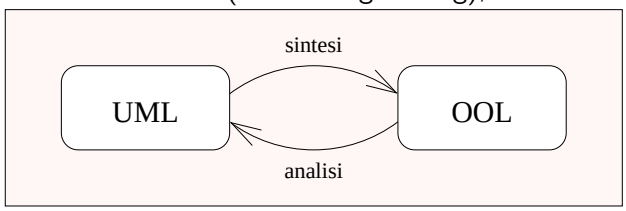
- Perché C++?
  - Acquisisce eredità di C ed è compatibile con programmi C esistenti.
  - È tuttavia cresciuto in modo abnorme ed è veramente difficile raggiungere piena competenza in tutti gli aspetti del linguaggio.
- Perché Java?
  - WWW e abilità di eseguire "web applet" su qualunque elaboratore e sistema operativo, attraverso un browser web.
  - Linguaggio ben progettato, con pochi costrutti.
  - Risolti i problemi della velocità di esecuzione.
  - Diffusamente utilizzato nell'ambito educativo (prima volta che un linguaggio di produzione reale coincide con un linguaggio di insegnamento).

Metodologie OOAD e UML

- Sono numerose e classificate in :
  - **Heavyweight**: le più datate e tradizionali, usate per grandi progetti software che coinvolgono centinaia di programmatori in anni di lavoro.
  - **Lightweight (agile)**: per progetti più piccoli.
- Ogni metodologia sviluppava una propria notazione grafica.
- Tutto ciò è cambiato con l'introduzione di un UML (Unified Modelling Language) [1990 - Rumbaugh, Jacobson, Booch].
- UML è progettato per discutere su OOAD e per descrivere progetti esistenti. Può essere anche visto come un vero e proprio linguaggio di programmazione.

UML

- Linguaggio i cui simboli sono diagrammi.
- Esiste una standardizzazione dei diagrammi "ben formati", analogamente ad una grammatica generativa.
- Utilizzato:
  - in fase di **concettualizzazione** – quando si devono esplicitare le specifiche di funzionamento di un sistema;
  - in fase di **analisi software** – quendo si devono descrivere i funzionamenti di un progetto software esistente (reverse engineering);



  - in fase di **sintesi software** – quando si deve progettare un sistema software, ed in questo caso UML è un vero e proprio linguaggio di programmazione.

Essenza di un oggetto

- Insieme di **attributi** (valore, stato interno, o qualunque altra cosa sia necessaria per il modello dell'oggetto).
- **Abilità** di modificare lo stato degli attributi.
- **Responsabilità** sotto forma di servizi offerti ad altri oggetti.
  Gli oggetti esterni non hanno conoscenza di come un oggetto realizza gli attributi internamente, ma piuttosto devono conoscere quali servizi un oggetto offre.

Classe

- È la descrizione di un insieme di oggetti che condividono attributi e comportamento comuni.
- Il concetto di classe è simile a quello del tipo di dato astratto nei linguaggi non OO.
- La definizione di una classe descrive tutti gli attributi comuni agli oggetti della classe, così come tutti i metodi che realizzano il comportamento comune degli oggetti della classe.
- Gli oggetti membri di una classe sono chiamati **istanze** di quella classe.
  I membri di una classe variano durante l'esecuzione di un programma.

Esempi

1. In un modello di un ROBOT un tipo di oggetto è ovviamente il *sensore*.
   La classe *Sensore* definisce le caratteristiche di base di un qualunque sensore, come la posizione nello spazio, il valore del trasduttore, un codice identificativo ed un insieme di servizi offerti per adempiere alle proprie responsabilità, i mezzi per accedere e modificare lo stato interno degli oggetti individuati dalla classe.
   Ogni oggetto istanza di *Sensore* avrà valori specifici per gli attributi descritti nella definizione di classe.
2. Una rappresentazione comune del colore è la RGB, dove ogni colore è specificato dalle componenti R (Red), G (Green), B (Blu).
   Una possibile definizione di una classe chiamata *Colore* dovrebbe fornire i mezzi per accedere o modificare il colore di un oggetto *Colore*.

Gerarchie

- In un sistema ad oggetti è tipico definire una classe basandosi su classi preesistenti o estendendo la descrizione in una classe di più alto livello oppure includendo la descrizione di un'altra classe all'interno della classe corrente.
- Si possono, per esempio, costruire le classi che descrivono sensori di pressione o di temperatura basandosi sulla generica classe *Sensore*.

  Aggiungere attributi e metodi $\longrightarrow$ sottoclasse
  Sottrarre attributi e metodi $\longrightarrow$ superclasse

- Vantaggi:
  se la stessa responsabilità è condivisa da più di una sottoclasse, porre il metodo che la realizza nella superclasse riduce le ripetizioni.

Peculiarità della programmazione OO 15 / 30

Astrazione con oggetti

- **Astrazione** è il meccanismo che permette di rappresentare una situazione complessa del mondo reale con un modello semplificato.
- Quella OO astrae il mondo reale usando oggetti e la loro interazione.

LP1 – Lezione 4 16 / 30

Classi incapsulate

- Nascondono la realizzazione di attributi e del comportamento di un oggetto al resto del mondo.
- Permettono a ciascun oggetto di essere indipendente.
- Vengono realizzate pensando alle proprietà degli oggetti che dovranno rappresentare:

  Attributi: possono essere strutture semplici, come una variabile, oppure complesse, come un altro oggetto.
  Comportamento: attività dell'oggetto così come viene vista all'esterno.
  Metodi: realizzano il comportamento dell'oggetto; sono operazioni o servizi offerti dall'oggetto all'esterno.
  Stato: riflette i valori correnti degli attributi che caratterizzano l'oggetto.

LP1 – Lezione 4 17 / 30

Scambio di messaggi

È il metodo con cui gli oggetti interagiscono.
Un oggetto invia messaggi ad un altro per:

- richiedere informazioni sullo stato interno,
- richiedere modifiche dello stato interno.

I messaggi possono essere

Sincroni: nei sistemi che simulano operazioni effettive oppure quando la risposta è necessaria per la continuazione del processo associato all'oggetto mittente.
Asincroni: l'oggetto mittente non attende la risposta, ma il suo processo prosegue in modo indipendente al processo del destinatario.

LP1 – Lezione 4 18 / 30

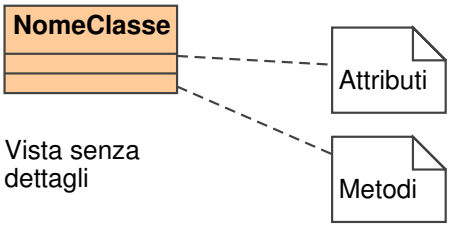
Ciclo di vita di oggetti

- Oggetti sono entità dinamiche; nascono durante l'esecuzione del programma come **istanze** di una certa classe, evolvono, poi muoiono.
- Alla generazione di una istanza, viene invocato un particolare metodo della classe genitrice chiamato **costruttore**. Esso è responsabile dello stato iniziale dell'istanza.
- Alla morte di un oggetto alcuni linguaggi richiedono l'invocazione esplicita di un metodo (dell'istanza) chiamato **distruttore**; altri linguaggi possiedono un meccanismo implicito di distruzione: un oggetto cessa di esistere quando non vi sono più riferimenti ad esso; a questo punto è il sistema stesso che riconosce i "cadaveri" e li elimina (**garbage collection**).

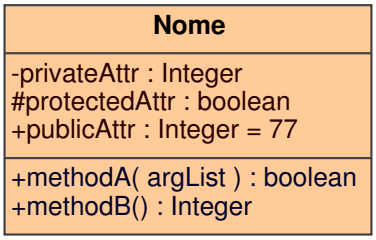
LP1 – Lezione 4 19 / 30

Sunto UML

Diagrammi di classe

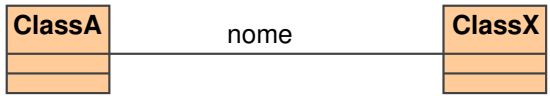


Vista senza dettagli



Vista dettagliata

Relazioni tra classi



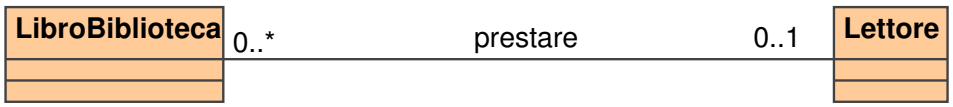
Associazione

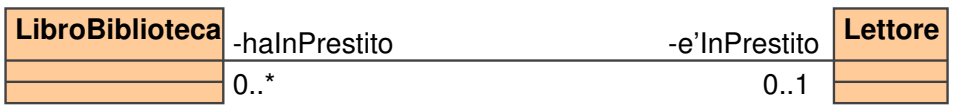
Aggregazione

Composizione

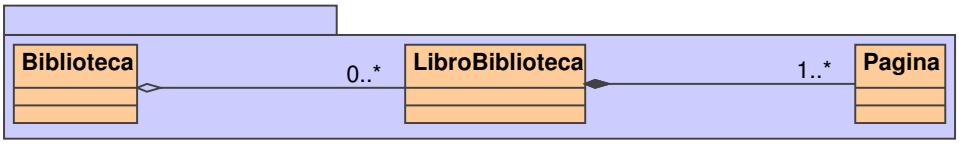
Associazioni

Attributo particolare in una classe, rappresentato in UML da una linea continua.



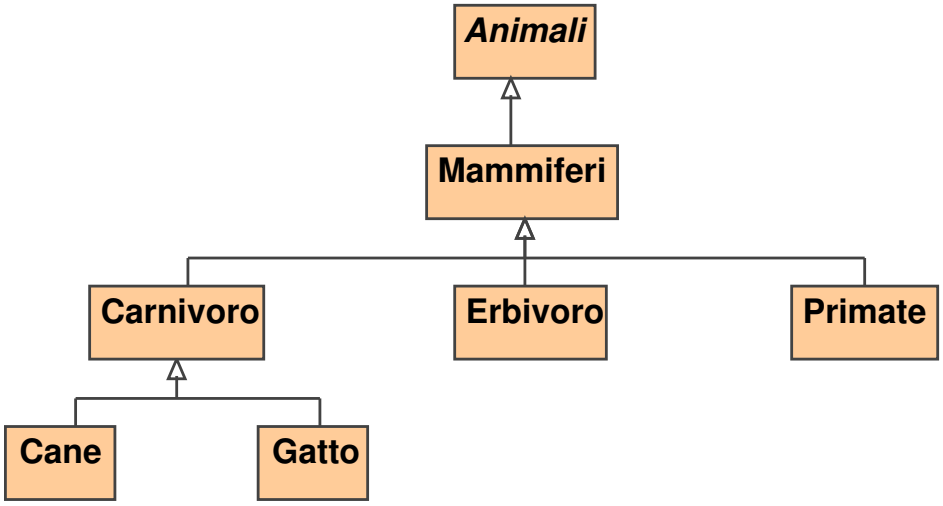


Composizioni e aggregazioni



- Relazione "ha_un".
- **Composizione**: oggetto composto esiste finché esistono le componenti (rombo pieno); inoltre, se non esiste più l'oggetto composto, non esisteranno più nemmeno le componenti.
- **Aggregazione**: oggetto aggregato esiste anche se non esistono gli aggreganti (rombo vuoto); inoltre, anche se non esiste più l'oggetto aggregato, gli aggreganti restano in vita.

Generalizzazioni



- Relazione "è_un".
- È realizzata mediante *ereditarietà*. Ogni sottoclasse è una specializzazione della superclasse ed eredita da essa (solo alcuni) attributi e/o metodi.

Note sulle Generalizzazioni

- Nota sull'ereditarietà multipla.
- Nota sulla relazione stessa.

  1. Billy è un PastoreTedesco
  2. PastoreTedesco è un Cane
  3. Cane è un Animale
  4. PastoreTedesco è una Razza
  5. Cane è una Specie

  Applicando la proprietà transitiva:

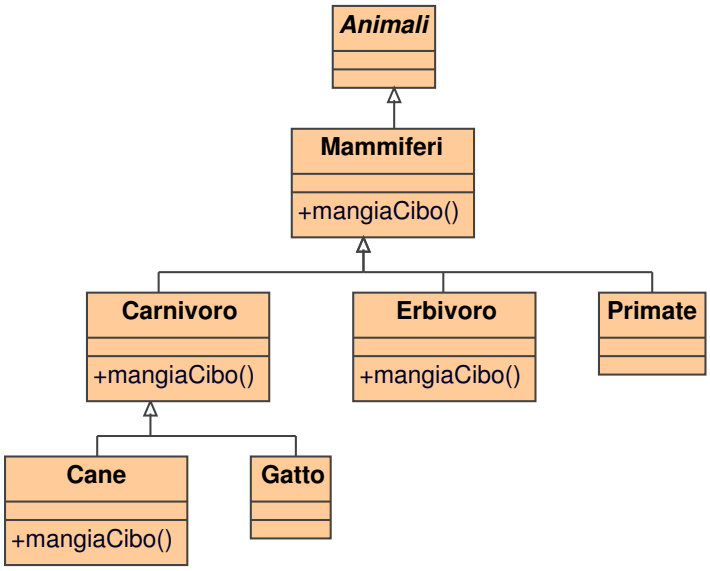
  - Billy è un Cane
  - PastoreTedesco è un Animale
  - Billy è un Animale

  Ma forse anche?

  - Billy è una Razza
  - PastoreTedesco è una Specie

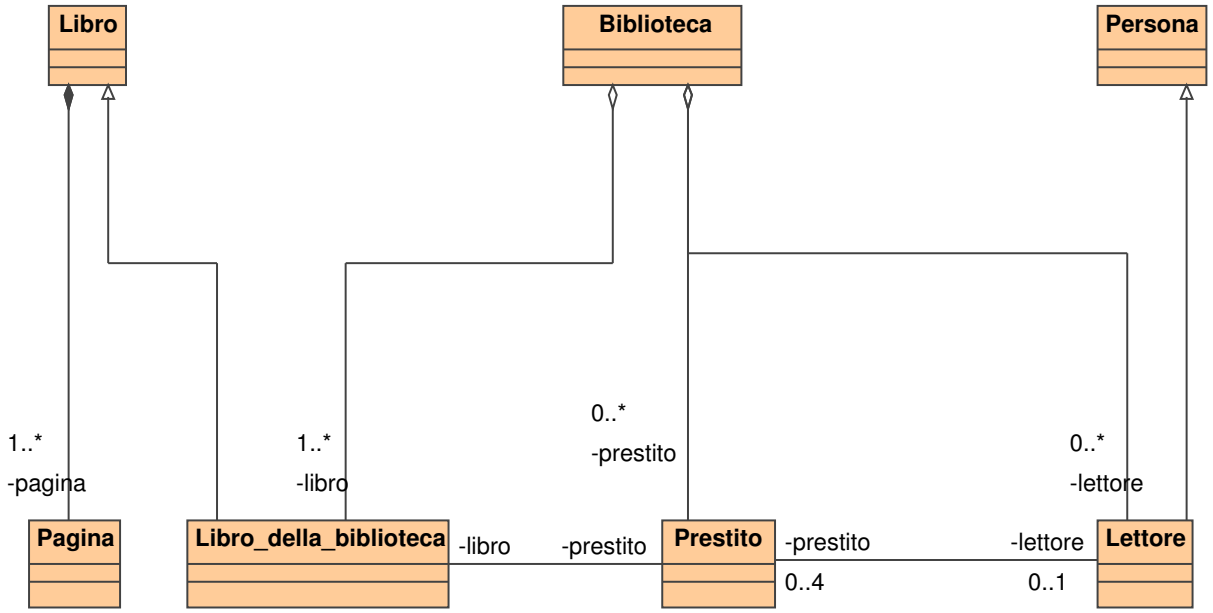
  Dov'è l'errore?

Polimorfismo



- Permette al mittente di ignorare l'identità (il tipo) del ricevente.
- È realizzato dal mix di overriding e upcasting (vedremo poi).

Un esempio



Altri concetti OO

Classi astratte

- Definiscono una interfaccia comune a tutte le sottoclassi, specificando solo i nomi (la firma) dei metodi che le sottoclassi "concrete" dovranno definire.
- Per esempio: tutti gli *`Animali`* devono avere un metodo `riproduci`, ma ogni sottoclasse definirà poi il metodo più appropriato.
- Poiché sono incomplete della realizzazione dei metodi, le classi astratte non possono avere istanze.
- Nei diagrammi UML hanno il nome in *`Italico`* (oppure `Sottolineato`).
- Richiamo sull'ereditarietà multipla in Java.

Visibilità delle componenti di una classe

- È la possibilità di una classe di vedere ed usare le risorse di un'altra classe. Livelli di visibilità:

  Private: attributi e metodi sono visibili solo a oggetti istanze della stessa classe. In UML: prefisso −.
  Package: attributi e metodi sono visibili ad un insieme specifico di classi. In UML: prefisso ~.
  Protected: oltre alla precedente, attributi e metodi sono visibili a tutti gli oggetti istanze della stessa classe o di una sua sottoclasse. In UML: prefisso #.
  Public: Attributi e metodi sono visibili a tutti gli altri oggetti. In UML: prefisso +.

- Normalmente attributi mai pubblici. Solo alcune operazioni sono pubbliche, quelle che forniscono servizi.
- Se si devono reperire o modificare le informazioni sullo stato interno di un oggetto, lo si fa utilizzando i metodi specifici (e visibili) dell'oggetto.

Bibliografia 30 / 30

Bibliografia

- Wampler B. E. *The essence of Object Oriented Programming with Java and UML*. Addison-Wesley. Cap. 1, 2.
- Fowler M. *UML distilled*. Addison-Wesley.