CORSO I.F.T.S.

"TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE"

Matricola 2014LA0033

DISPENSE DIDATTICHE
MODULO DI "PROGETTAZIONE SOFTWARE"

Dott. Imad Zaza

Lezione del 16/07/2014

Ristorante

I camerieri, dotati di dispositivi palmari, hanno il compito di memorizzare le ordinazioni dei var clienti, e di stilare il conto per i vari tavoli

Un responsabile delle prenotazioni ha il compito di prenotare i tavoli per i clienti che lo chiedono e di inserire i loro dati anagrafici (solo per nuovi clienti)

E' in facolta di questio ultimi di poter scegliere i tavoli per fumatori o non fumatori

Ristorante

Si osservi che è previsto che le prenotazioni possano essere disdette, ma solo se non state ancora effetuate ordinazioni.

All'arrivo dei clienti, il responsabile delle prenotazioni assegna loro uno o più cameriri

Si vuole infine dare la possibilità al responsabile del personale di sapere quanti tavoli i vari cameieri hanno già dsservito nel giorno corrente, al fine di permettere un bilanciamento corretto del carido di lavoro tra di essi.

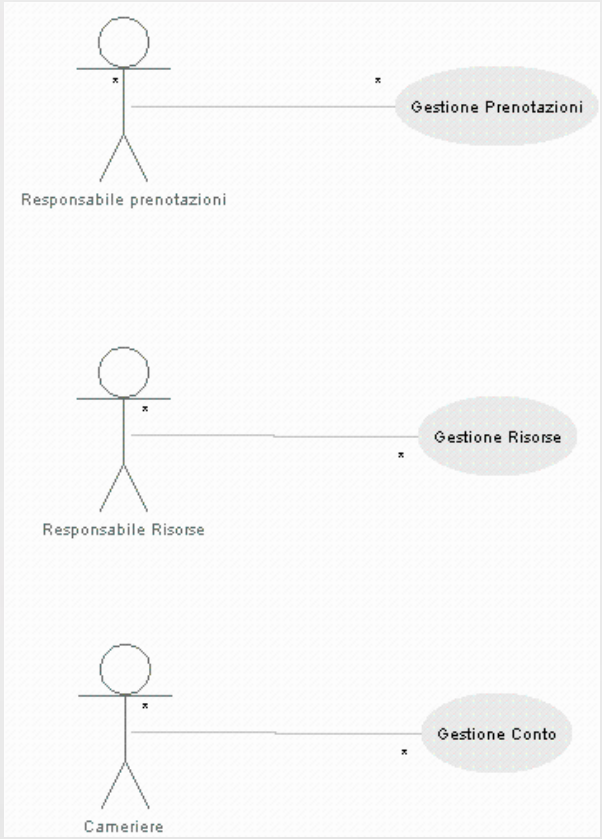
Attori

- Camerieri
- Responsabile delle prenotazioni
- Responsabile del personale

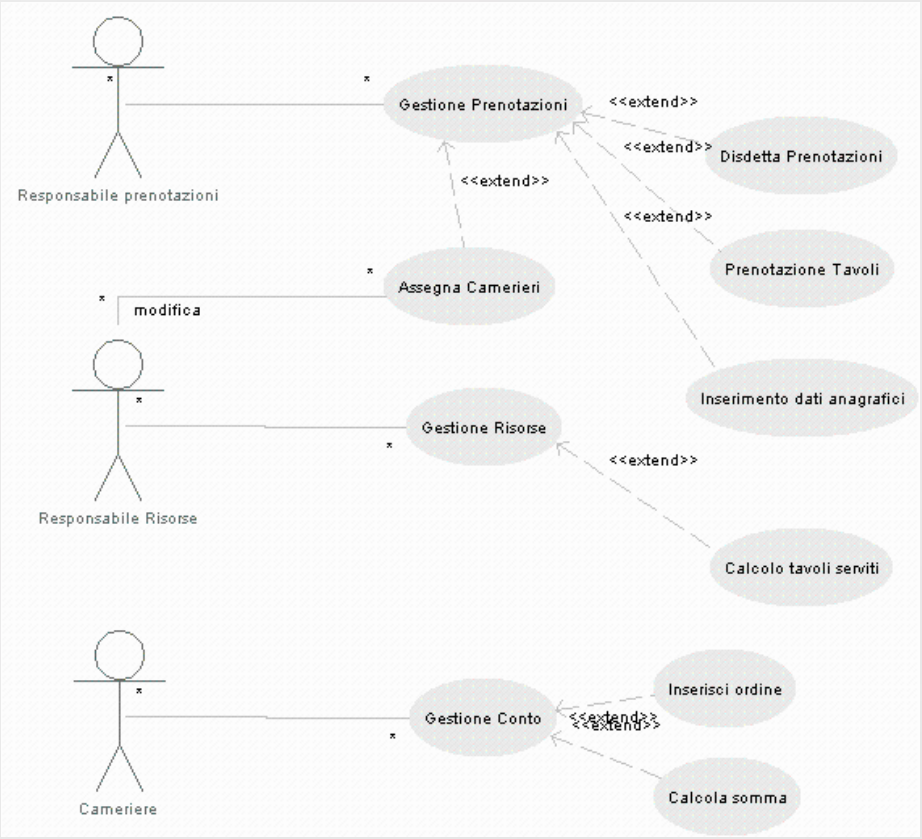
Use Case

- Stilare il conto
- Prenotare i tavoli
- Inserire i loro dati anagrafici
- Prenotazioni possano essere disdetta
- Assegna loro uno o più camerieri
- Sapere quanti tavoli i vari camerieri hanno gia servito

Use case diagram L0



Use Case diagram L1



Catena di officine

I dipendenti delle officine hanno il compito di registrare i dati dei veicoli in ingresso, di interrogare l'archivio delle riparazioni da effettuare, di effettuarle e registrarne la terminazione, consegnando i veicoli riparati ai clienti. Tuttavia, per riparazioni particolarmente complesse, quest'ultimo compito viene lasciato ai direttori, che provvedono al rilascio di una particolare garanzia ai clienti. Infine i direttori devono avere la possibilità di interrogare ed eventualmente modificare i dati personali dei propri dipendenti, mentre un ufficio di marketing, occupandosi delle varie comunicazioni ai clienti, deve poter accedere ai loro dati.

Attori

Dipendenti

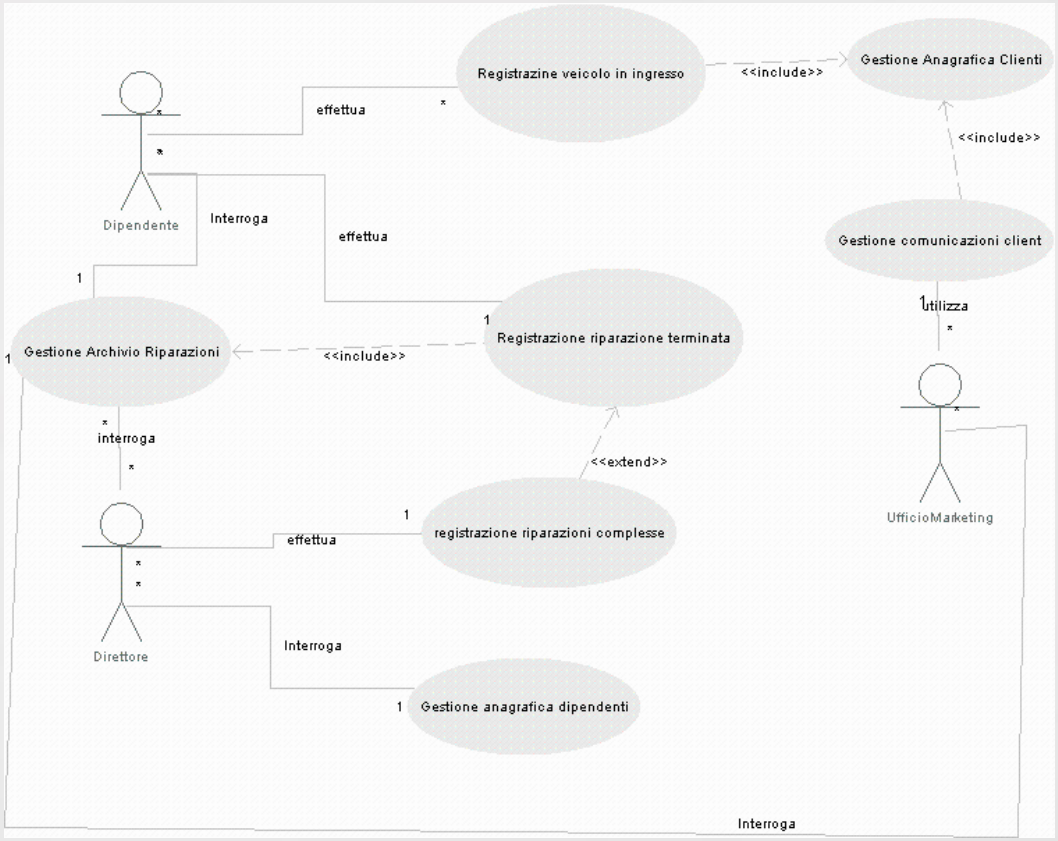
Direttori

Ufficio marketing

Use Case

- Registrare i dati dei veicoli in ingresso
- Interrogare l’archivio delle riparazioni
- Registrarne la terminazione
- Devono avere la possibilità di interrogare ed eventualmente modificare i dati personali dei propri dipendenti
- Deve poter accedere ai loro dati

Use Case Diagram



Campionato di calcio

Si vuole progettare un sistema per la gestione di un campionato di calcio. Il sistema deve consentire la creazione del calendario delle partite e la disegnazione degli arbitri (da parte della FIGC).

L'arbitro avrà il compito di memorizzare nel sistema, a fine gara, il risultato finale. Ogni squadra ha un proprio allenatore che decide quali giocatore convocare per le varie partite.

E facoltà della presidenza delle squadra acquistare e vendere giocatori o cambiare allenatore .

Infine, La FIGC viole poter stampare la classifica e la schedina relativa alle diverse giornate di un particolare girone.

Attori

- Arbitri
- FIGC
- Allenatore
- Presidenza

Use Case

- Creazione del calendario
- Designazione arbitri
- Memorizzare sistema
- Risultato finale
- Decide quali giocatori convocare
- Acquistare e vendere giocatori
- Cambiare allenatore
- Stampare classifica e schedina

Use Case Diagram

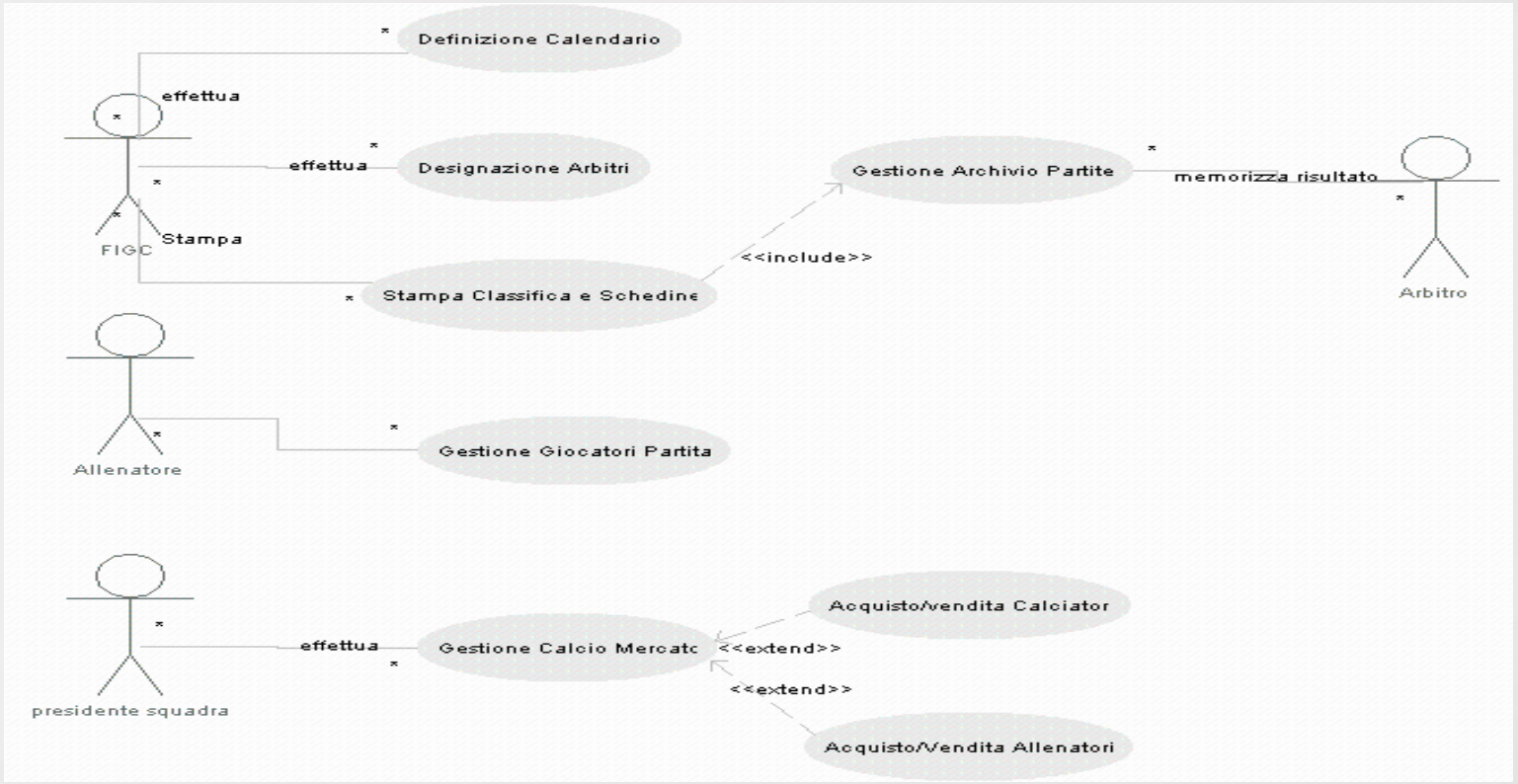


Diagramma delle classi

- è il caposaldo dell'object oriented
- rappresenta le classi di oggetti del sistema con i loro attributi e operazioni
- mostra le relazioni tra le classi (associazioni, aggregazioni e gerarchie di specializzazione/generalizzazione)
- può essere utilizzato a diversi livelli di dettaglio (in analisi e in progettazione)

Programmazione ad Oggetti: Brevissimo

- Per gestire la complessità del sistema da realizzare occorre considerare solamente le *proprietà essenziali dei moduli software che si devono implementare* (*criterio di astrazione*) in una classe di problema ben specificata (*dominio del problema*).

- Si devono omettere i dettagli non necessari alla risoluzione del problema nel suo dominio.

Rappresentazione del dato

- L’astrazione può essere effettuata a due livelli
- ASTRAZIONE SUI DATI
  - consiste nell’astrarre le entità (oggetti) costituenti il sistema, descritte in termini di una struttura dati e delle operazioni possibili su di essa.
- ASTRAZIONE SUL CONTROLLO
  - consiste nell'astrarre una data funzionalità dai dettagli della sua implementazione

Rappresentazione del dato

- Nei linguaggi procedurali, il tipo di dato è l'insieme dei valori che può assumere una variabile dello stesso tipo.

- Nella programmazione ad oggetti, il *tipo di dato astratto (Abstract Data Type o ADT)* estende questa definizione includendo anche tutte le possibili operazioni che possono essere effettuate su quel determinato tipo. La struttura dati è pertanto incapsulata nelle operazioni su di essa definite

Oggetti

- Nella Programmazione ad Oggetti (in inglese Object Oriented Programming o OOP) le unità base per la costruzione del sistema sono gli *oggetti*, che sono istanze di *classi*.
- Una classe rappresenta l'astrazione di oggetti aventi le stesse caratteristiche nel dominio del problema da risolvere.
- Es: la classe delle automobili; la classe dei radar;

Classe

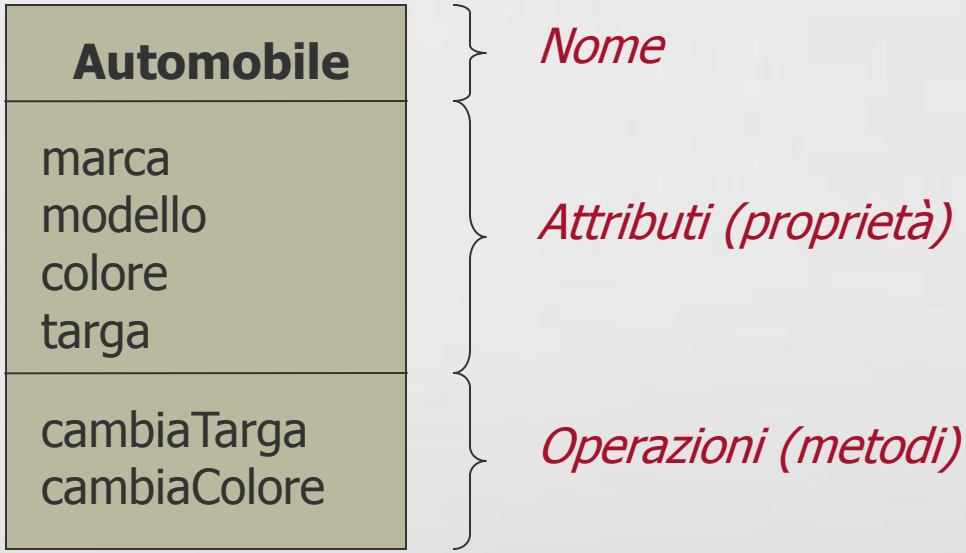
Le caratteristiche che definiscono una classe sono:

- gli *attributi* o *dati membro* , che sono le variabili associate ad una classe
- I *metodi* o *funzioni membro*, che sono le procedure o servizi che la classe *mette a disposizione* (chiamate anche interfaccia o protocollo).

Le classi pertanto definiscono gli ADT e da esse si *istanziano* (cioè si *creano*) gli oggetti

Class diagram

- Rappresentazione di una classe in UML:



Class diagram

- Nome: inizia con una lettera maiuscola, non è sottolineato e non contiene underscore, In grassetto e centrato
  - In corsivo se la classe è astratta
- Attributi: proprietà i cui valori identificano un oggetto istanza della classe e ne costituiscono lo stato; iniziano con una lettera minuscola
- Operazioni: insieme di funzionalità che esprimono il comportamento di un oggetto, ovvero ciò che ogni oggetto di questa classe può fare
- Visibilità
  - - privata
  - + pubblica
  - # protetta

Attributi ed Operazioni

- Attributi:
  - visibilità / nome : tipo [molteplicità] = valore-default {proprietà}
  - Derivati: Sono attributi non strettamente necessari che possono essere derivati da altri attributi della classe (/Nome Attr)

- Operazioni:
  - visibilità nome (lista_parametri): return_type {proprietà}

Vincoli

- Il campo {proprietà} può essere convenientemente usato per esprimerevincoli sugli attributi e sulle operazioni
  - Attributi: valori
  - Operazioni: precondizioni/postcondizioni
- Un vincolo va inteso come un contratto che deve essere rispettato in fase di implementazione!

Caretteristiche

I linguaggi per la programmazione ad oggetti hanno le seguenti caratteristiche in comune:

- Incapsulamento
- Ereditarietà
- Composizione
- Polimorfismo
- Binding Dinamico

Incapsulamento

- L’incapsulamento è la proprietà per cui le struttura interna dell’oggetto è nascosta agli altri oggetti.
- La comunicazione tra oggetti avviene tramite l’invio di ***messaggi***. L’implementazione dei messaggi avviene tramite le chiamate a funzione dei metodi degli oggetti. L’incapsulamento fa sì che le operazioni dell’oggetto non possono essere in nessun modo influenzate da altri oggetti.

Eriditarietà

- L’ereditarietà (inheritance) è il meccanismo per cui un oggetto acquisisce le proprietà di un altro oggetto in base al concetto di classificazione gerarchica.
- La classe che eredita le proprietà si chiama sottoclasse
- La classe che fornisce le proprie caratteristiche si chiama superclasse
- La sottoclasse “eredita” le caratteristiche della superclasse (secondo determinate modalità)
- Una classe può essere sottoclasse di differenti superclassi (eredità multipla o multiple inheritance)

Ereditarietà (sottoclasse)

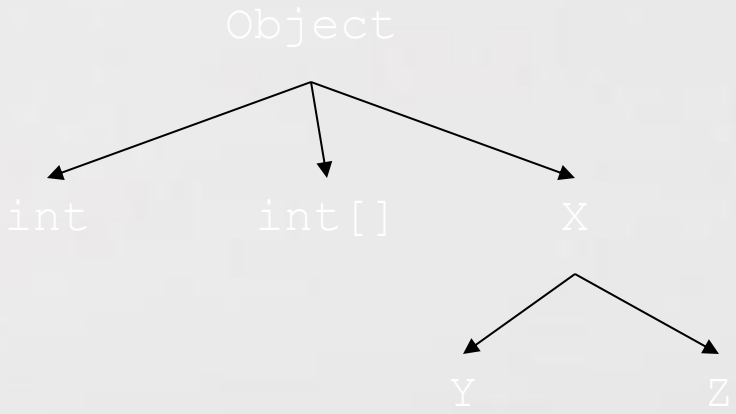
Relazione tra implementazioni basata sull'idea che una sottoclasse reimpiega, in tutto o in parte, il codice di un classe già implementata.

```
class Body {
   private String name;
   public String getname()
    { return name;}
}
```

```
class CelestialBody
      extends Body {
      private CelestialBody
       orbits;
}
```

La gerarchia delle classi

Poiché ogni classe può estendere al più una classe, si genera una gerarchia ad albero, alla radice della quale è la classe `Object`:

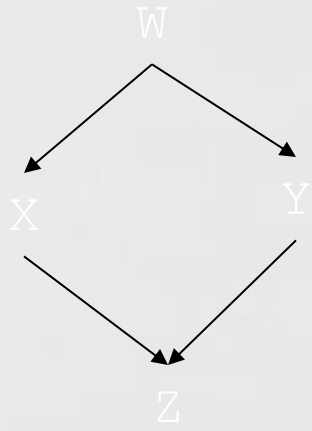


`class A {…}` equivale a `class A extends Object {…}`.

Il grafo dei sottotipi

D’altra parte un’ interfaccia può estendere più interfacce:

```
interface W {…}
interface X extends W {…}
interface Y extends W {…}
interface Z extends X, Y {…}
```



Ridefinizione (overriding)

Una classe Y che estende una classe X può anche ridefinire un metodo della classe X conservandone la segnatura:

```
class X {
   private int n;
   public int f(int m)
    {return n+m;}
   …
}
```

```
class Y extends X {
   public int f(int m)
    {return n += m;}
   …
}
```

In Java: sottoclasse = sottotipo

Poiché in Java le classi sono tipi, la relazione di sottoclasse si identifica con quella di sottotipo

```
class A {…}
class B extends A {…}
…
B obj1 = new B();
A ob2 = obj1; // sussunzione
```

Abbinamento dinamico dei metodi

Se `obj` è un riferimento ad un oggetto di classe Y che estende X, allora per eseguire `obj.f()` si procede:

1. si cerca il corpo di `f()` in Y: lo si esegue se trovato, altrimenti
2. si cerca il corpo di `f()` nella superclasse X: lo si esegue se trovato, altrimenti si cerca nella superaclasse di X. Ecc.

La parola chiave `super`

L'uso di `super` consente di chiamare la versione della sopraclasse di un metodo ridefinito:

```
class A { void f() {} ..}
class B extends A {
 void f() {}; // ridefinizione di f()
 void g() { … super.f(); …} }
B b = new B();
b.g(); // la chiamata di f() esegue il metodo
    // f() di A
```

Nota: questo è il solo caso in cui il tipo governa l'abbinamento del metodo

Abbinamento dinamico dei metodi (2)

Nell'invocazione di un metodo conta la classe cui l'oggetto appartiene e non il tipo del riferimento attraverso cui si accede all'oggetto:

```
class A {
  public void f() {System.out.print('A');}
class B extends A {
  public void f() {System.out.print('B');}
A obj = new B();
obj.f(); // produce la stampa di 'B'
```

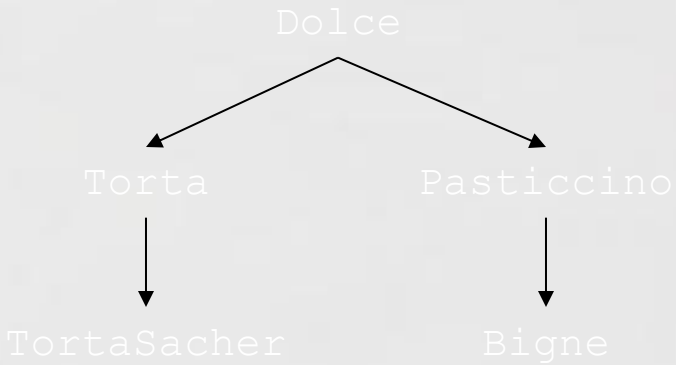
Sovraccarico (overloading)

Quando nella stessa classe un metodo ha più definizioni con diversa segnatura

```
void f(A a);
void f(A a, D d); // diverso numero di param.
void f(B b, D d); // diversi tipi dei param.
```

Quale metodo si applica?

```
class Dolce {}
class Torta extends Dolce {}
class TortaSacher extends Torta {}
class Pasticcino extends Dolce {}
class Bigne extends Pasticcino {}
```




```
void sceglie(Dolce d, Pasticcino p)       // (a)
void sceglie(Torta t, Dolce d)            // (b)
void sceglie(TartaSacher ts, Pasticcino p)  // (c)
sceglie(dolce, pasticcino)       // forma (a)
sceglie(tortasacher, dolce)      // forma (b)
sceglie(tortasacher, bigne)      // forma (c)
sceglie(torta, pasticcino)       // illegale
```

Poblemi: casting del valore ritornato

La necessità di mantenere il tipo dei metodi ridefiniti (overridden) obbliga all'uso del casting:

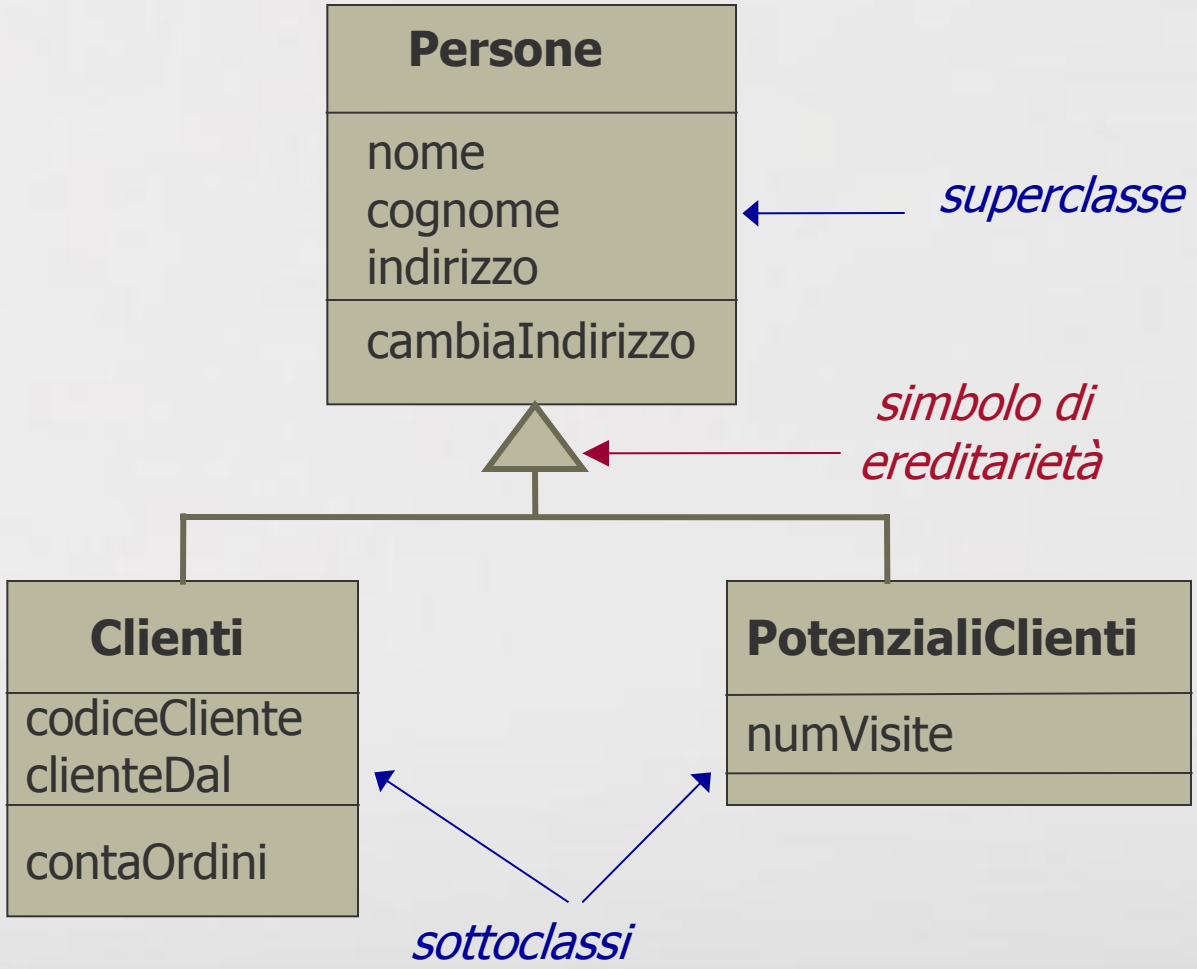
```
protected Object clone()
// produce una copia dell'oggetto, ma non degli
// oggetti riferiti al suo inetrno
class A {
   // public A clone() {…} // illegale
   public Object clone() {…} …}
A b = (A)a.clone(); // supposto a di classe A.
```

Problemi: metodi binari

Un metodo è *binario* se tra i suoi parametri ve n'è uno il cui tipo è lo stesso dell'oggetto cui il metodo appartiene.

```
class A {public boolean eq(A other){…} …}
class B extends A {
   // public boolean eq(B other) {…} // illegale
   public boolean eq(A other) {…} …}
B b1 = new B(); b2 = new B();
b1.eq((A)b2); // casting del parametro
```

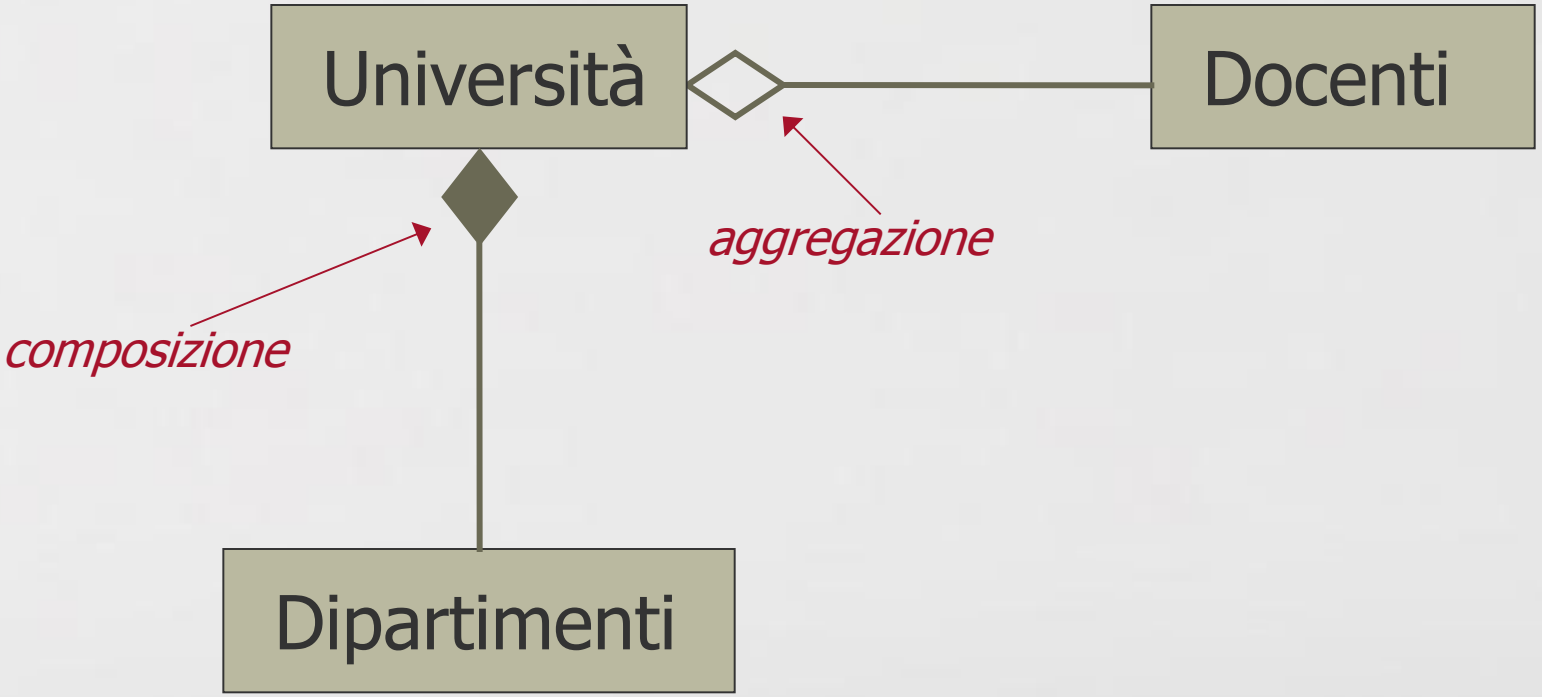
Class diagram: ereditarietà



Composizione

- Vi è la possibilità di definire un oggetto come composto da oggetti diversi.
- Una struttura in cui un oggetto è composto da più oggetti si chiama ***composizione o aggregazione o APO (a-part-of).***
- Es: La classe auto è composta dalle classi motore, scocca, ruota , etc. Ognuna di queste classi può eventualmente essere scomposta in altre APO a seconda della natura del problema da risolvere.

Class diagram: aggregazione/composizione



Class diagram: associazioni

- *Associazione:* correlazione fra classi; nel diagramma è una linea continua fra due classi, con esplicita semantica nei due sensi
- *Molteplicità:* numero di oggetti che partecipano all'associazione. Esempi di molteplicità sono:

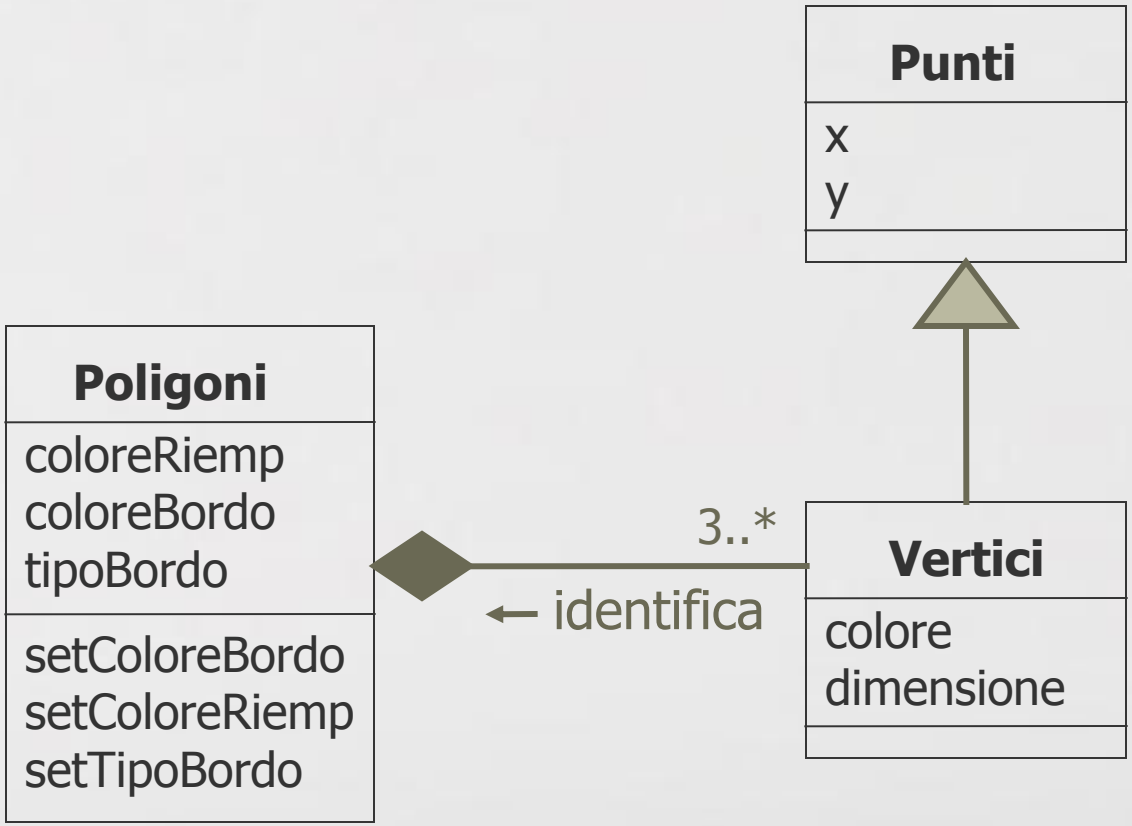
1	**Esattamente una istanza**
0..*	**Nessun limite al numero di istanze**
1..*	**Almeno una istanza**
n..m	**Da n a m istanze**

Class diagram: aggregazione

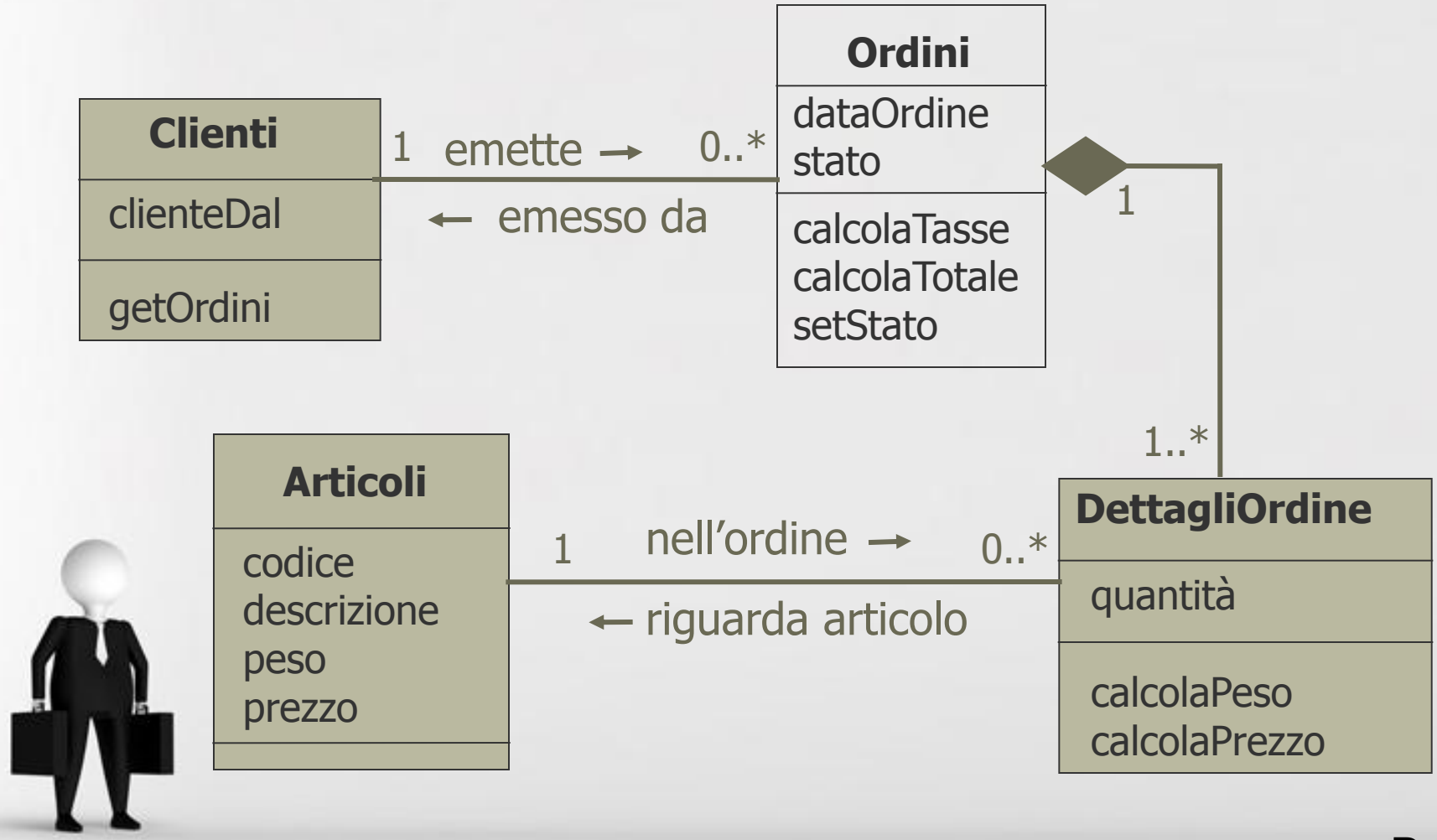
- *Aggregazione:* tipo particolare di associazione; esprime concetto "*è parte di* " (*part of*), che si ha quando un insieme è relazionato con le sue parti.

- Si rappresenta con un diamante dalla parte della classe che e' il contenitore

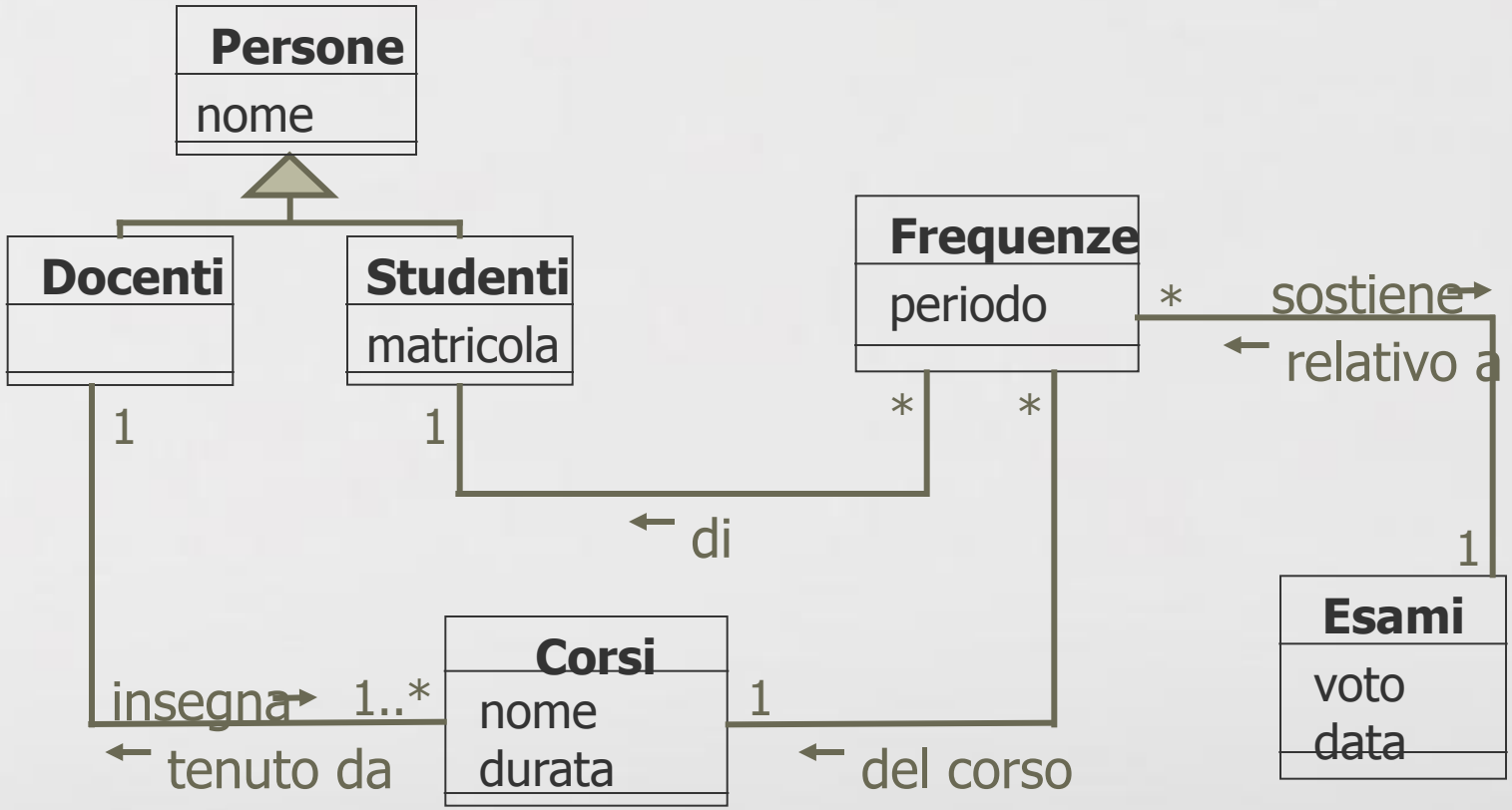
Class diagram: esempio



Class diagram: associazioni



Class diagram: esempio



Polimorfismo

- Il *polimorfismo* è la capacità che hanno *oggetti diversi*, ma correlati in quanto derivanti da una superclasse comune, di *rispondere in maniera diversa ad uno stesso messaggio*.
- Es: La classe rettangolo e esagono derivano dalla classe poligono. Il calcolo dell'area è un metodo che ha senso nella classe poligono, ma ha un diverso comportamento per il rettangolo e per l'esagono.

Che cosa è il polimorfismo?

Polimorfismo: quando certi valori o certe variabili possono avere più di un tipo.

Varie forme di polimorfismo:

• **parametrico**: lo stesso codice si comporta uniformemente su valori (dati) di diverso tipo

• **inclusione**: si basa su una relazione di specializzazione

• **sovraccarico**: quando una funzione opera in modo differente su valori di tipo differente

• **coercizione**: quando un valore di un tipo viene visto (e talvolta trasformato) come valore di un altro tipo.

Sottotipo ed Ereditarietà

- Interfacce
  - La vista esteriore degli oggetti di una classe
- Sottotipo
  - Relazione di specializzazione tra interfacce
- Implementazione
  - La rappresentazione interna degli oggetti di una classe
- Ereditarietà
  - Relazione tra implementazioni

Che cosa è un'interfaccia?

E' un'astrazione che elenca alcuni metodi pubblici posseduti da un oggetto

Può vedersi come una classe sottospecificata, di cui si omettono la parte privata, i campi ed il corpo dei metodi pubblici

L'interfaccia di un oggetto è il suo *tipo*

Interfaccia: esempio

```
interface Point {
   double getx(); void setx(double x);
   double gety(); void sety(double y);
   void move(double dx, double dy);
}
```

Tutti i metodi di un'interfaccia sono `public`; in Java le interfacce non possono contenere campi privati, ma solo campi `public static final`.

Sottotipo

Relazione di specializzazione tra tipi:

B sottotipo di A se ogni oggetto di tipo B può essere impiegato dove ci si aspetterebbe un oggetto di tipo A

```
interface ColoredPoint extends Point {
    Color getcolor();
    void setcolor(Color c);
}
```

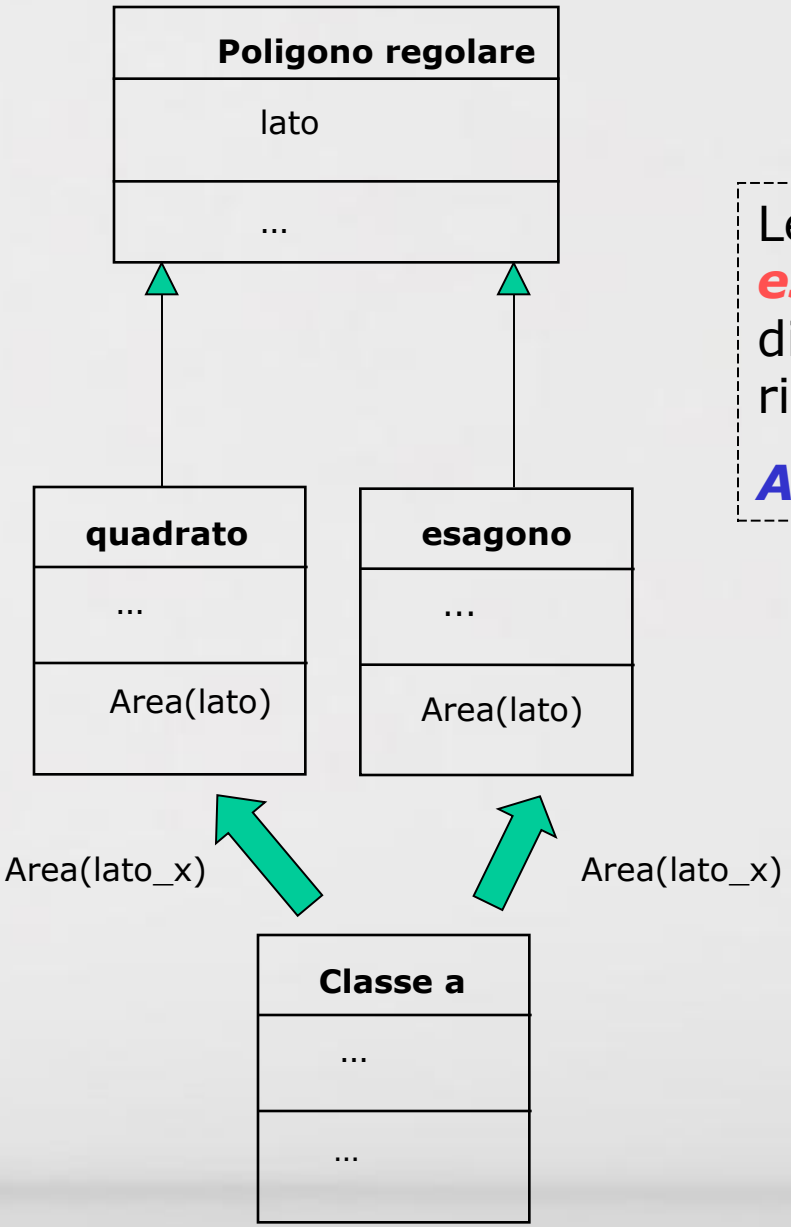
Una classe implementa un'interfaccia

Una classe specifica ed estende i metodi elencati nell'interfaccia

```
class MyPoint implements Point {
  private double x; private double y;
  public getx() { return x;}
  public void setx(double val){x = val;}
  ...
}
```

Una classe può implementare più interfacce.

Esempio



Le ***classi quadrato*** ed ***esagono*** si comportano differentemente a fronte della ricezione del messaggio

Area(lato_x)

OT: Binding Dinamico

Il binding dinamico (o binding tardivo o late binding) di un linguaggio orientato agli oggetti è l'abilità di determinare la classe dell'oggetto a run-time e allocare memoria per esso.

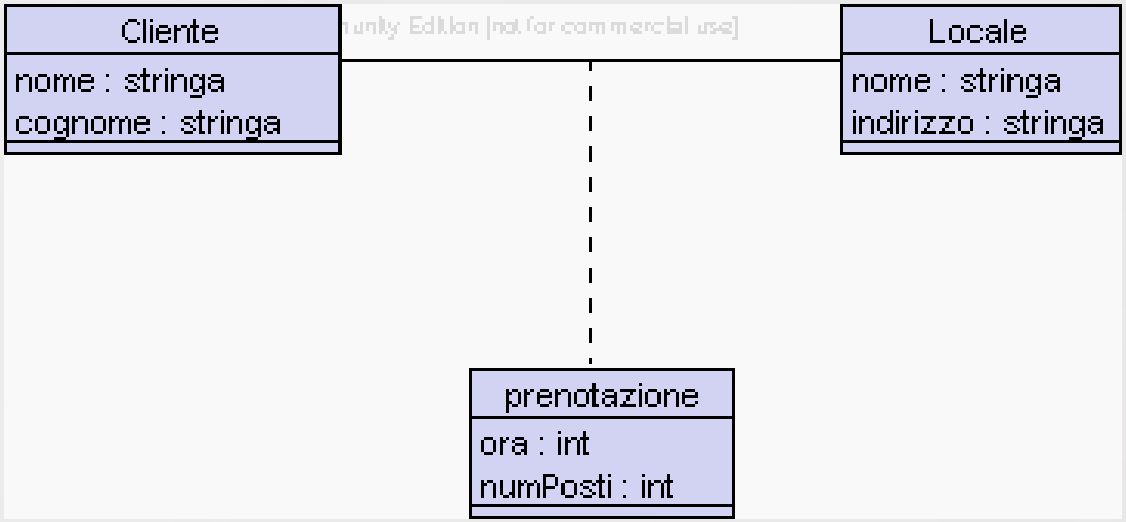
Il binding dinamico è trasparente allo sviluppatore. In altre parole, chi sviluppa un programma non deve usare nessuna istruzione particolare affinché si esegua il binding dinamico.

Il binding dinamico ha la sua controparte nel binding statico o precoce (early binding), dove l'allocazione è effettuata dal compilatore.

Locale

Si vogliono modellare i clienti che prenotano presso un locale. Dei clienti interessa il nome e il cognome.
Del locale la via e il nome. Della prenotazione l'ora e il numero di posti da prenotare.

Soluzione


Cliente
nome : stringa
cognome : stringa
Locale
nome : stringa
indirizzo : stringa
prenotazione
ora : int
numPosti : int

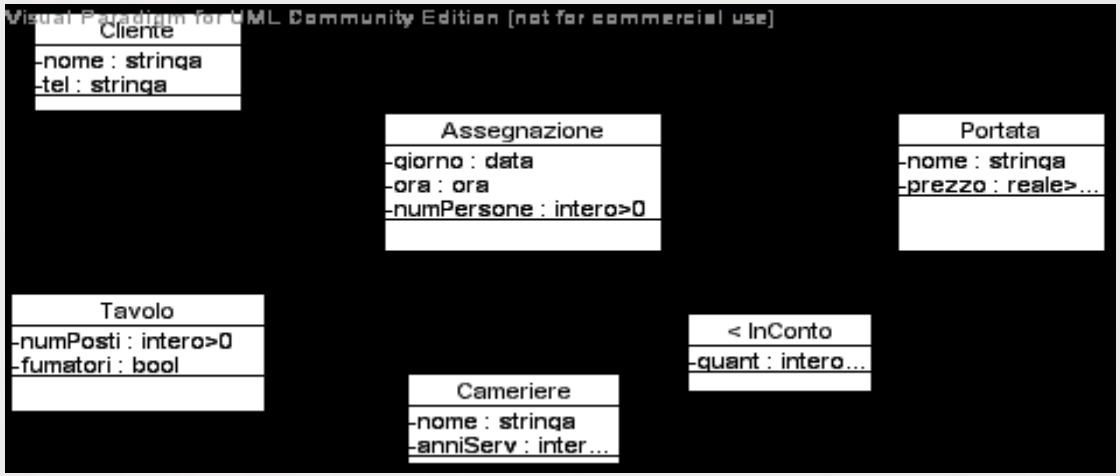
Azienda

L’azienda X è costituita da diversi dipartimenti, ad ognuno dei quali afferisce un certo insieme di impiegati. Ogni impiegato (del quale interessa il nome, l'età, lo stipendio) afferisce esattamente ad un dipartimento. Dei dipartimenti interessa il nome, il numero di telefono, la data di afferenza di ognuno degli impiegati che vi lavorano, ed il direttore. Gli impiegati partecipano a vari progetti aziendali, dei quali interessa il nome ed il budget.

Ristorante

In un ristorante sono entità di interesse i clienti, i tavoli (con il relativo numero di posti), le prenotazioni (effettuate dai clienti per un certo giorno ed ora, ed un certo numero di persone) alle quali viene assegnato uno o più tavoli, i camerieri (che servono i clienti al tavolo) ed i conti relativi ai vari tavoli (contenenti i prezzi delle singole portate ordinate, e le loro quantità).
Dei clienti interessa il nome e numero di telefono, mentre dei camerieri interessa nome e anni di servizio.
Infine delle portate interessa il nome ed il prezzo unitario.

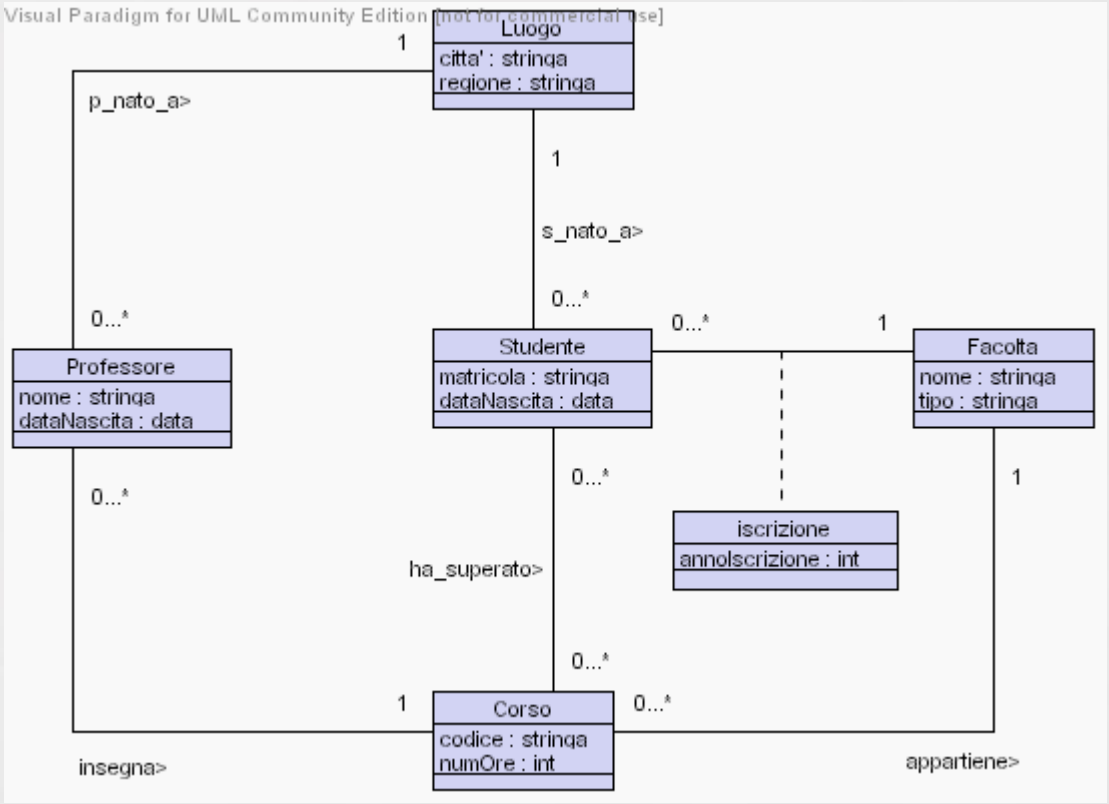
Soluzione


Visual Paradigm for UML Community Edition [not for commercial use]
Cliente
-nome : stringa
-tel : stringa
Assegnazione
-giorno : data
-ora : ora
-numPersone : intero>0
Portata
-nome : stringa
-prezzo : reale>...
Tavolo
-numPosti : intero>0
-fumatori : bool
< InConto
-quant : intero...
Cameriere
-nome : stringa
-anniServ : inter...

Università

Degli studenti interessa il numero di matricola, la data di nascita, il luogo di nascita (città e regione), la facoltà in cui è iscritto (con l'anno di iscrizione), e i corsi superati. Dei professori interessa il nome, la data di nascita, il luogo di nascita e il corso insegnato. Delle facoltà interessa il nome ed il tipo (scientifica, letteraria, ecc..). Dei corsi interessa il codice, il numero di ore di lezione, e la facoltà a cui appartiene.

Soluzione


Visual Paradigm for UML Community Edition [not for commercial use]
Luogo
citta' : stringa
regione : stringa
1
p_nato_a>
1
s_nato_a>
0...*
0...*
Professore
nome : stringa
dataNascita : data
0...*
Studente
matricola : stringa
dataNascita : data
0...*
1
Facolta
nome : stringa
tipo : stringa
0...*
iscrizione
annoIscrizione : int
ha_superato>
0...*
1
1
Corso
codice : stringa
numOre : int
0...*
insegna>
appartiene>