
MAJOR BIT ACADEMY

MAJOR BIT ACADEMY

OBIETTIVI:

1. Sviluppare la capacità di trovare soluzioni a macroproblemi esistenti nel perimetro di interesse del mondo reale, applicando la metodologia di analisi e disegno ad OGGETTI .
2. Sviluppare le abilità di formalizzazione della soluzione identificata utilizzando la notazione grafica standard UML (MetaModeling, Class Diagram, Use Case, ecc)

CU3 : *"Principi OOA - OOP e UML"*

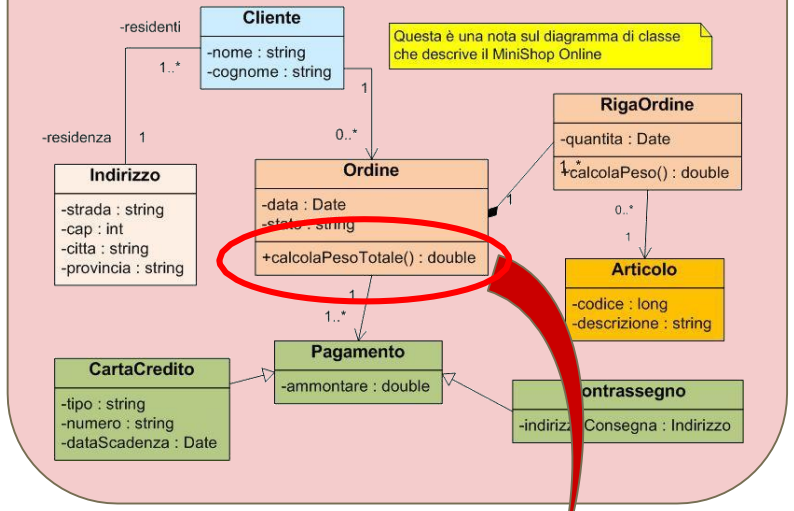
Analisi e Programmazione ad Oggetti

Progettazione di una applicazione

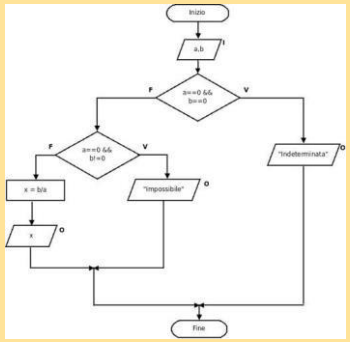
PROBLEMI NEL MONDO REALE

Analisi e Progettazione del MacroProblema (Scomposizione in CLASSI)

ARCHITETTURA AD OGGETTI

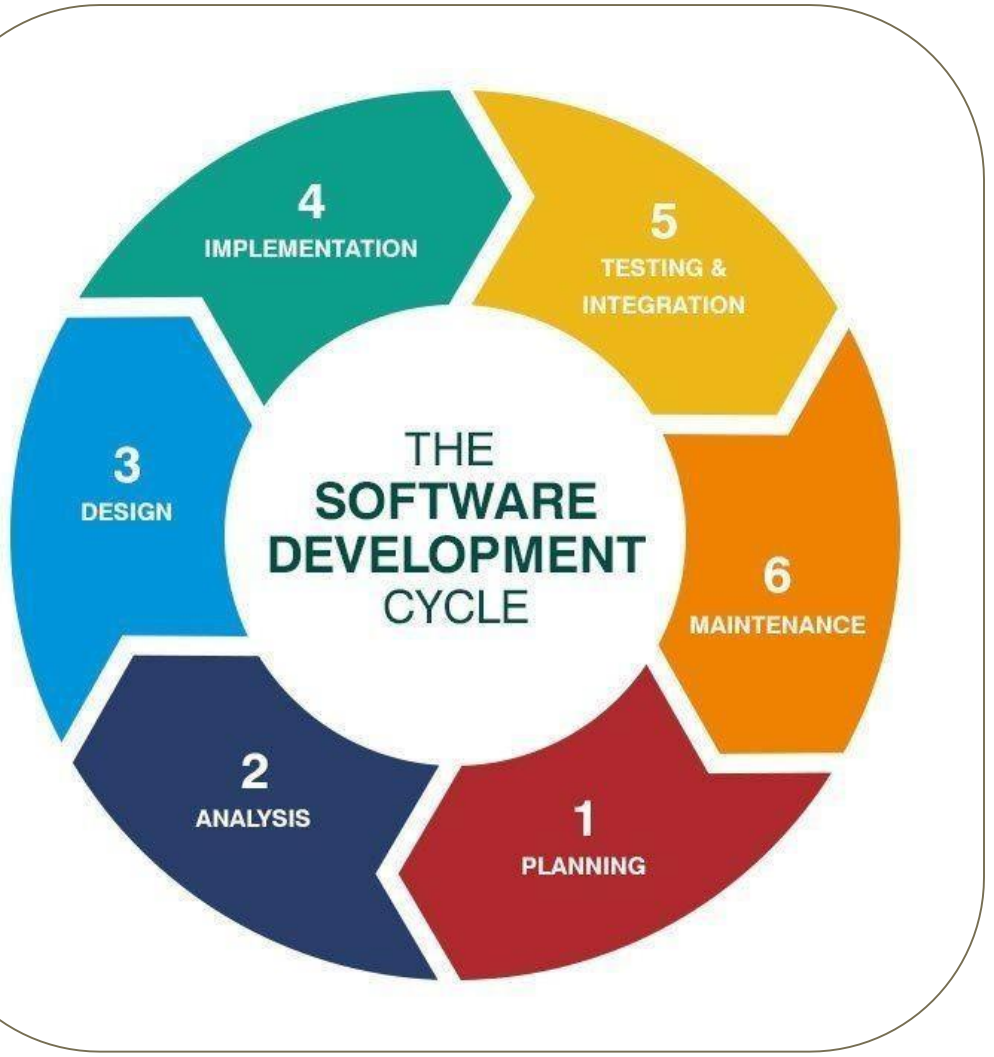


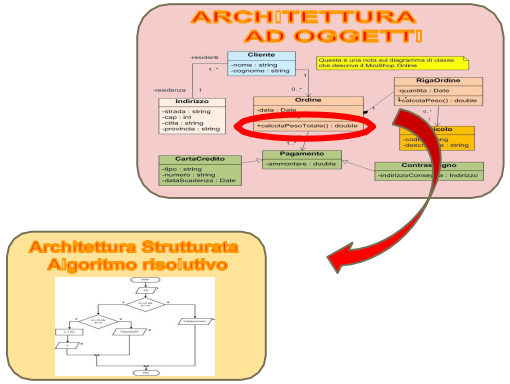
Architettura Strutturata Algoritmo risolutivo



Analisi e Progettazione di un MicroProblema

Ciclo di vita del Software





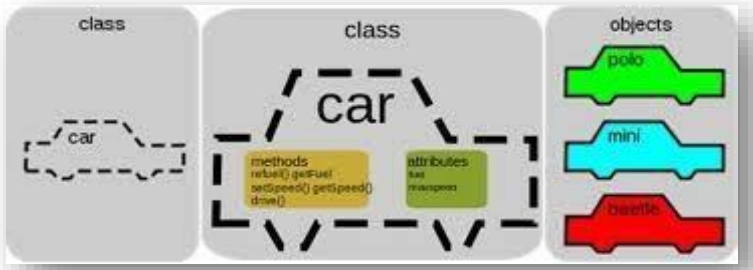
Principi e concetti OO: *definizione Classe e Oggetto*

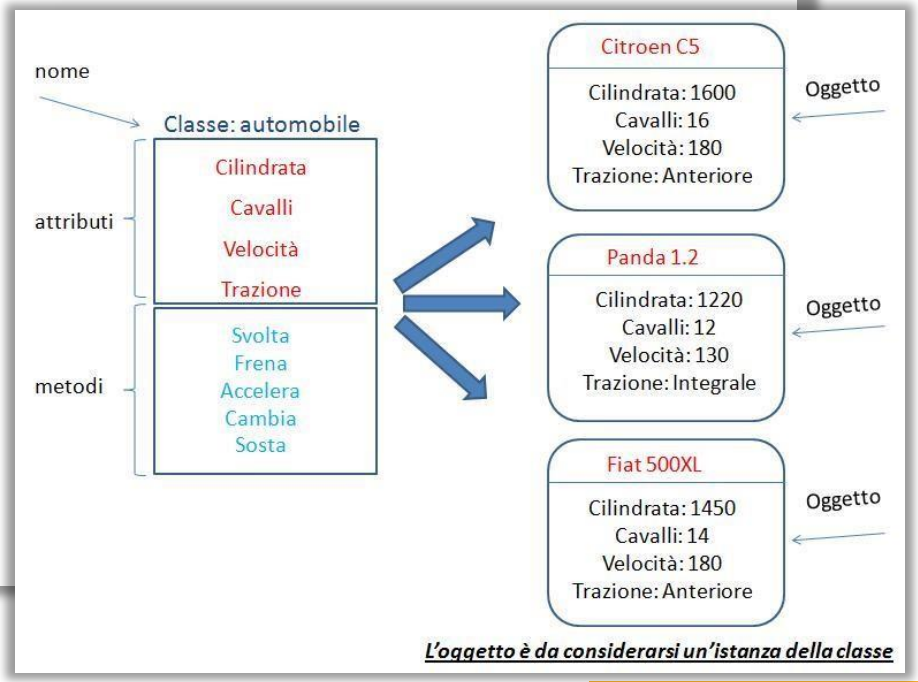
Una **classe** è la definizione di un'entità, descritta come un'astrazione di un oggetto. E' composta da *attributi* e *metodi*.

La classe definisce al proprio interno:

- gli **attributi** (chiamati anche *variabili d'istanza*) i quali definiscono le caratteristiche dell'entità;
- i **metodi** (descritti da blocchi di codice) che definiscono il comportamento dell'entità.

Un **oggetto** è l'istanza di una classe, cioè la concretizzazione della stessa.
È un modo di incapsulare dati (attributi) regolandone l'accesso tramite opportuni metodi.

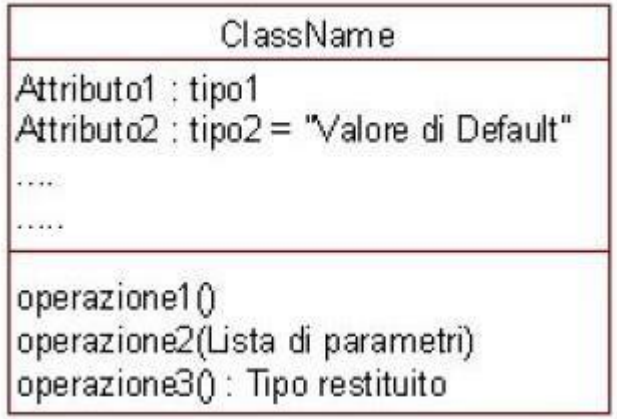




Progettazione OO: *Tipo astratto e CLASSE*

- Un tipo di dato astratto è in estrema sintesi il concetto di tipo di variabile applicato ad un oggetto
- Una classe è un modo di rappresentare un tipo di dato astratto
- Una classe è un'astrazione di tipizzazione di un oggetto
- Una classe è un tipo di oggetto
- Un tipo di dato astratto è quindi un modulo che incapsula sia la definizione di un tipo, la cui struttura risulta invisibile all'esterno, sia l'insieme delle operazioni che permettono di manipolare gli oggetti (istanze) di quel tipo

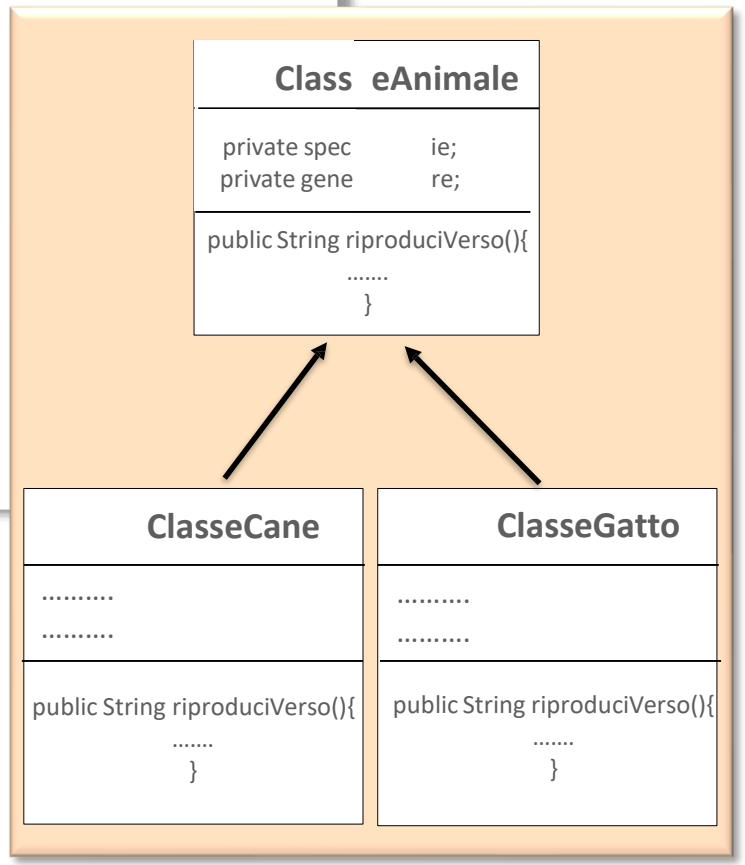
Esempio di tipo di dati astratto per gestire un conto corrente:



Progettazione OO: *vantaggi OOP*

Object - Oriented Programming

- OOP → divide il problema in oggetti separati che realizzano azioni relazionandosi con altri oggetti
- Vantaggi proncipali di OOP:
  - Dati e operazioni incapsulati in un oggetto
  - Quando viene creato un nuovo tipo di oggetto, non è necessario modificare le implementazioni precedenti
  - Piuttosto, il nuovo oggetto eredita alcune caratteristiche precedenti



Progettazione OO: *Modello*

Un *modello* è una rappresentazione semplificata della realtà che contiene informazioni ottenute focalizzando l'attenzione su alcuni aspetti cruciali e ignorando alcuni dettagli

4

L'**analisi** si occupa di:
- Definire la soluzione giusta per il problema giusto

La **progettazione** si occupa di
- Descrivere (anticipare) una soluzione al problema mediante un modello

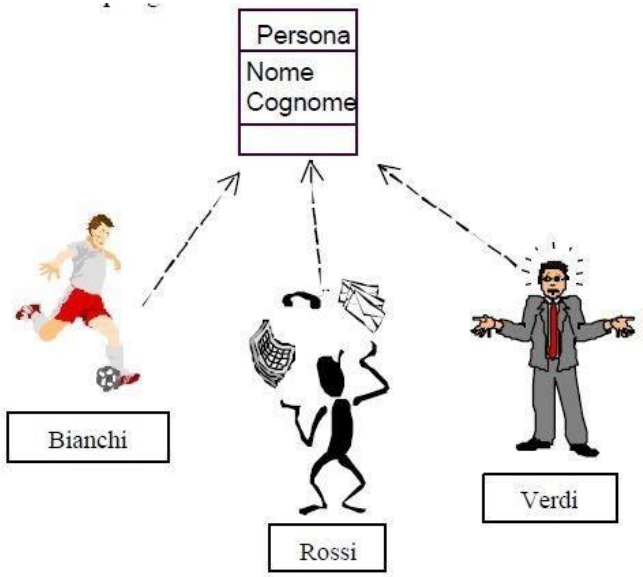


Figura 1 - Classificazione

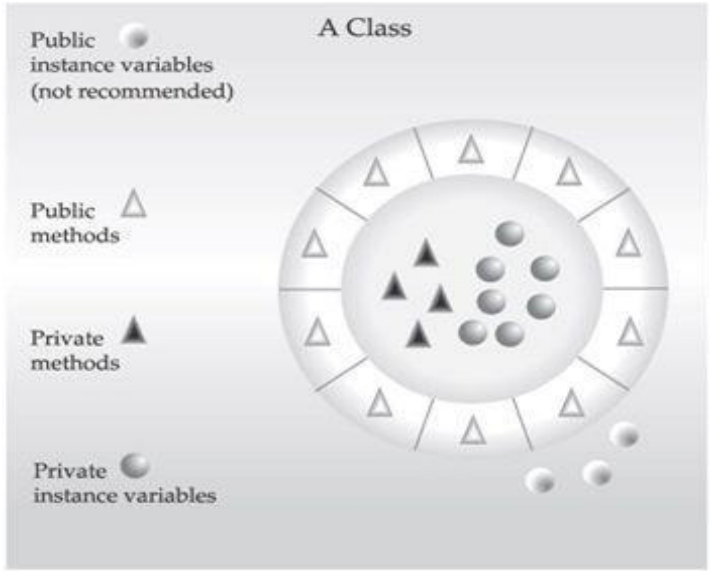
Un modello object-oriented [**Booch94**] è caratterizzato dai seguenti elementi costitutivi:

- astrazione di classificazione, la capacità di descrivere un oggetto mettendo a fuoco le caratteristiche più importanti e ignorando le altre, per mezzo del costrutto di classe;
- incapsulamento, la possibilità di nascondere i dettagli della implementazione di un oggetto;
- modularità, la possibilità di decomporre un sistema in un insieme di elementi caratterizzati da un basso grado di accoppiamento e da un elevata coesione interna;
- ereditarietà, la capacità di specificare una gerarchia fra le classi definite;
- aggregazione, la possibilità di definire un oggetto come composto da altri oggetti.

Principi OO

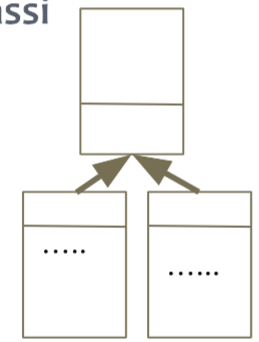
INCAPSULAMENTO

Incapsulare (impacchettare) attributi e metodi all'interno di una classe e usare opportunamente metodi privati e pubblici consentirà di proteggere le informazioni da accessi inopportuni



EREDITARIETA'

Consente la creazione di classificazioni gerarchiche definendo la Superclasse e le sottoclassi



OVERRIDING:

ridefinire il comportamento di un metodo in una sottoclasse mantenendo la stessa dichiarazione del metodo

Polimorfismo dinamico:

capacità di selezionare ed eseguire automaticamente a run time il metodo specifico della sottoclasse

POLIMORFISMO

OVERLOADING:

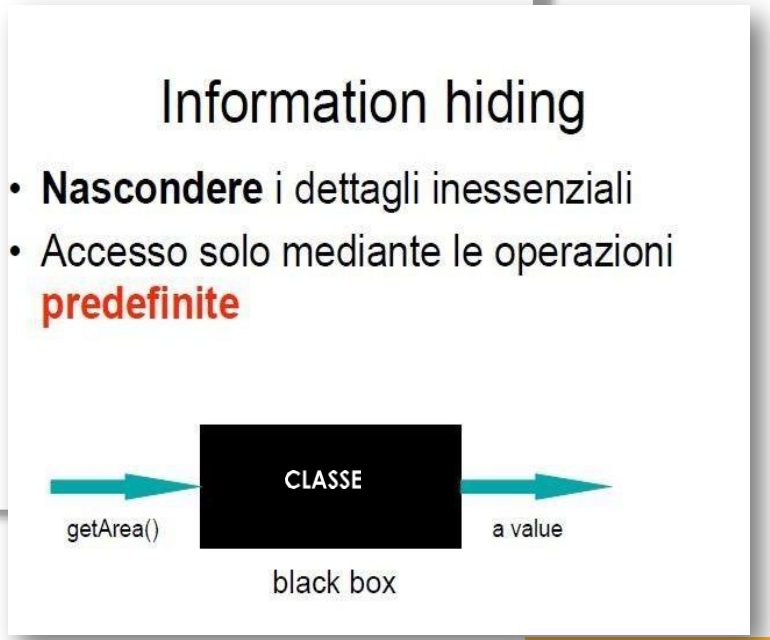
Due o più metodi nella stessa classe che condividono lo stesso nome

Principi OO: *definizioni*

L'**incapsulamento**: indica la proprietà degli oggetti di mantenere al loro interno attributi che i metodi (cioè **stato** e **comportamento)** dell'oggetto.

Attributi e metodi sono quindi incapsulati all'interno dell'oggetto.

L'information hiding è un concetto strettamente collegato all'incapsulamento ed è la capacità di nascondere (rendere **privati**) alcuni attributi e metodi all'interno dell'oggetto, cioè resi invisibili agli altri oggetti.



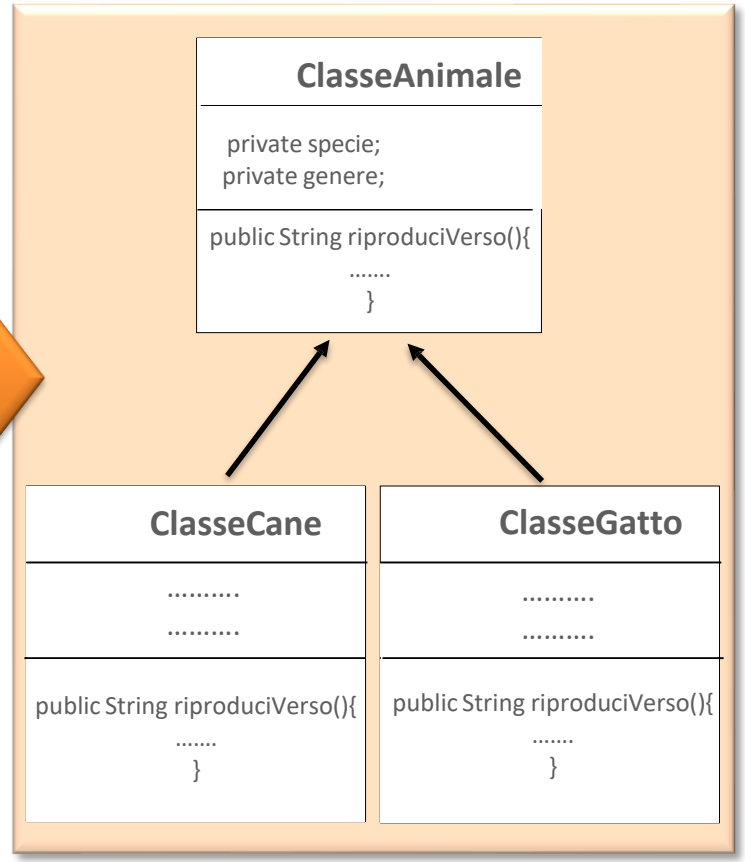
Principi OO: *definizioni*

Ereditarietà: è quel principio che permette di estendere una superclasse.

Le classi figlie:

- erediteranno attributi e metodi della classe padre;
  - potranno ridefinire i metodi ereditati;
- potranno aggiungere funzionalità proprie (attributi e metodi);

N.B. (gli attributi e metodi "private" verranno ugualmente ereditati anche se non visibili.)



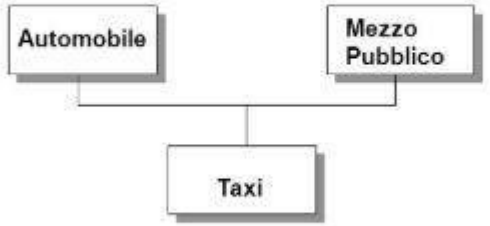
L'ereditarietà consiste quindi nella creazione di *sottoclassi* a partire da classi più generali, cioè nell'estensione di classi esistenti per crearne altre più *specializzate*.

Una ragione importante per usare l'ereditarietà è la possibilità di **riutilizzare il codice.**

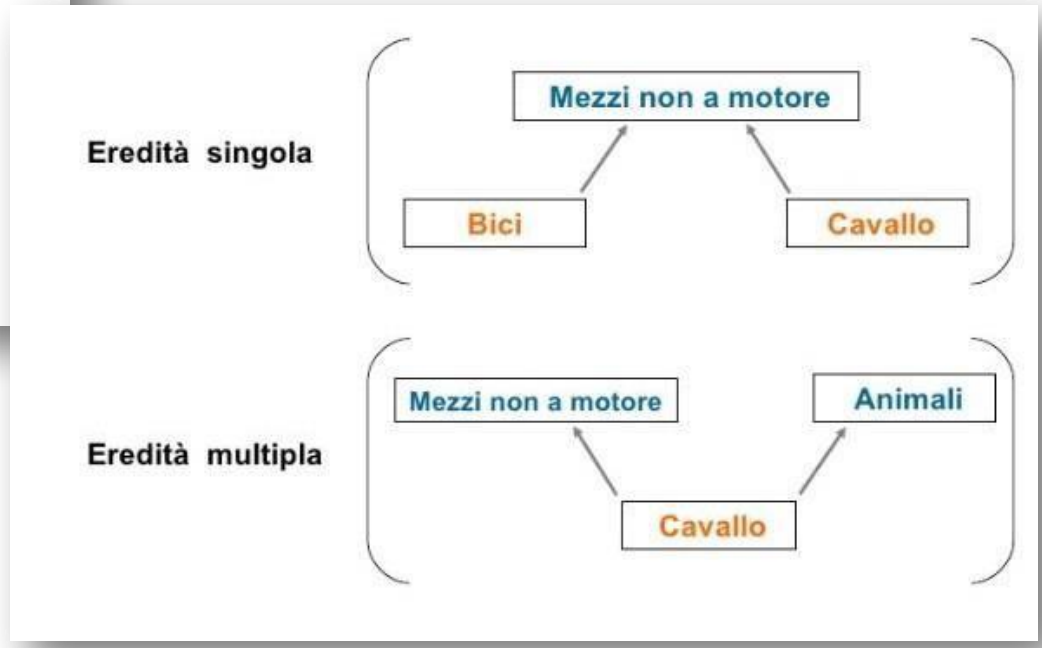
Principi OO e concetti: ereditarietà multipla

EREDITARIETÀ MULTIPLA

- Meccanismo che consente di derivare sottoclassi da due o più classi



- Il problema degli omonimi
- Java non implementa ereditarietà multipla



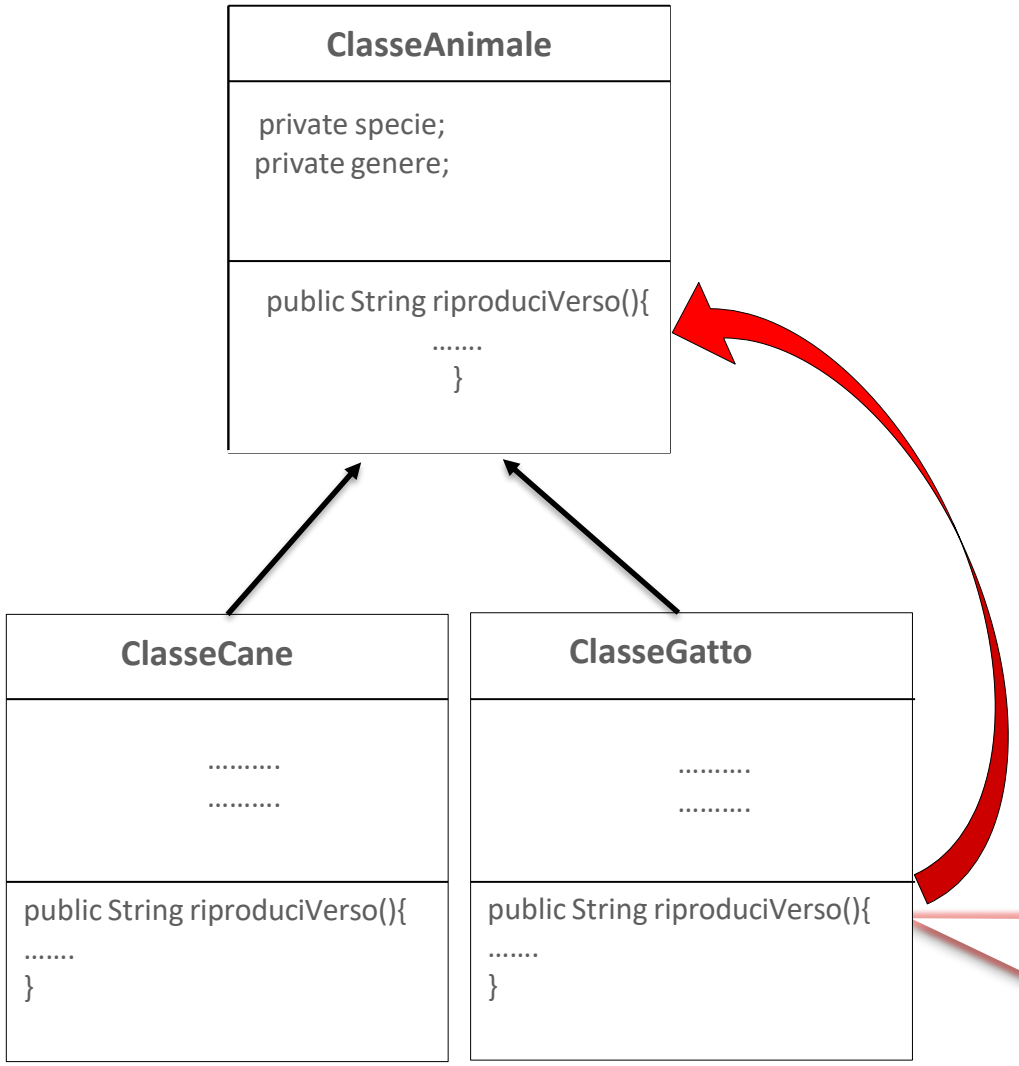
Principi OO: *definizioni*

Polimorfismo: è quel principio che permette di assumere più forme.

Il **Polimorfismo**
si divide in:

OVERRIDING **OVERLOADING**

Principi e concetti OO : *Polimorfismo e Overriding*



OVERRIDING:

È una proprietà del polimorfismo ed è strettamente collegato al concetto di ereditarietà.

Le classi figlie ereditano i metodi (stessa dichiarazione: nome e parametri) della superclasse ridefinendo il comportamento cioè implementando una diversa elaborazione al loro interno (diverso body).

OVERRIDING

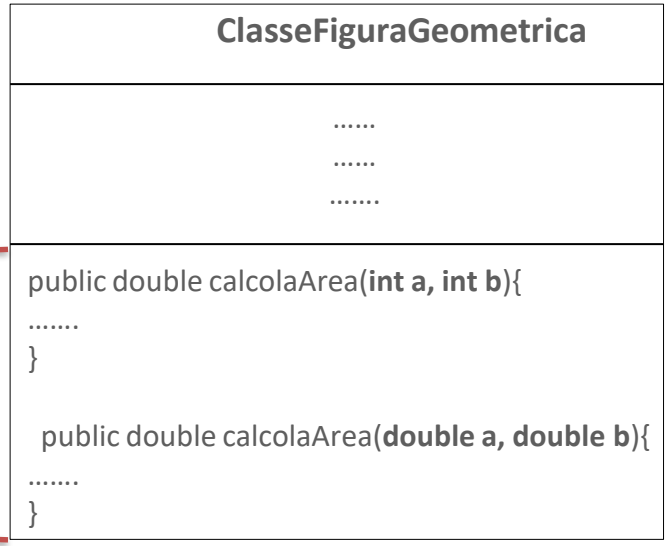
Stessa dichiarazione

- Stesso modificatore d'accesso
- Stesso tipo del valore di ritorno
- Stesso nome
- Stessi parametri

Body diverso

Principi e concetti OO : *Polimorfismo e Overloading*

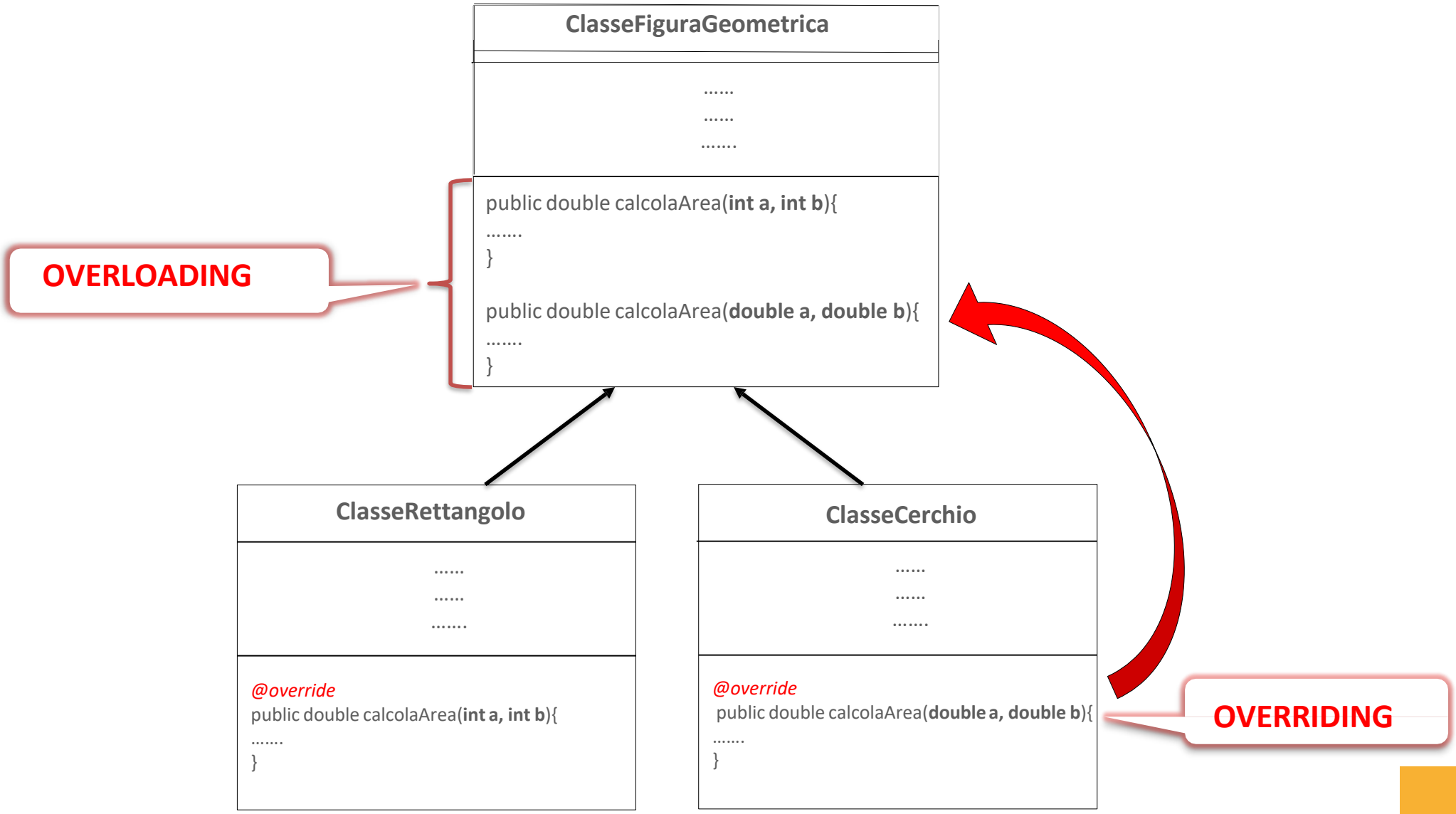
OVERLOADING
- Stesso nome
- Parametri diversi
- Body diverso



OVERLOADING:

È una proprietà del polimorfismo in cui sarà possibile avere più metodi con lo stesso nome all'interno di una classe. Tali metodi dovranno avere però, diversi parametri in input e dunque avranno comportamenti diversi in quanto diversa sarà l'elaborazione dei dati al loro interno (implementazione).

Principi e concetti OO : *esempi Polimorfismo*



- Modificatori di accesso: *tipi di visibilità*

Tipi di visibilità

+	(visibilità pubblica): ogni elemento che può accedere alla classe può anche accedere a ogni suo membro con visibilità pubblica
-	(visibilità privata): solo le operazioni della classe possono accedere ai membri con visibilità privata
#	(visibilità protetta): solo le operazioni appartenenti alla classe o ai suoi discendenti possono accedere ai membri con visibilità protetta
~	(visibilità package): ogni elemento nello stesso package della classe (o suo sottopackage annidato) può accedere ai membri della classe con visibilità package

Modificatori di accesso: *convenzioni*

Convenzioni sui nomi

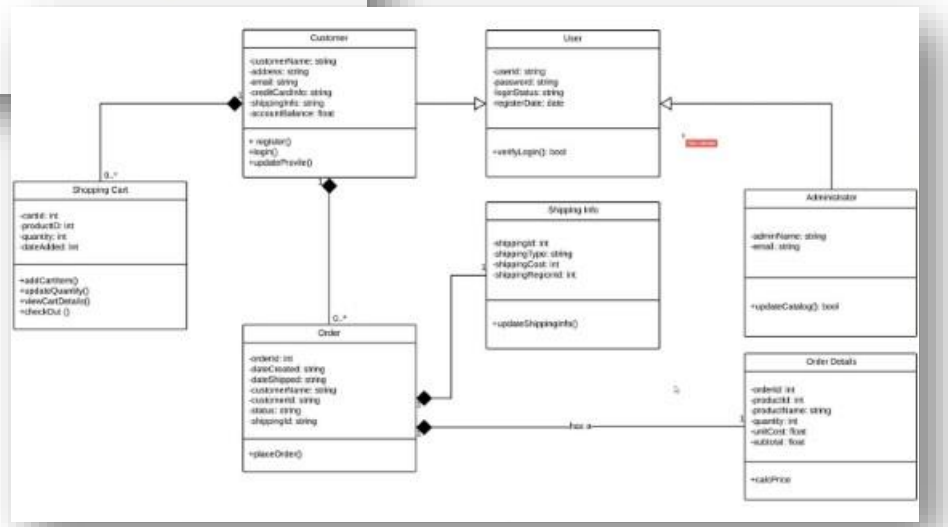
- **Classe**: lettera maiuscola iniziale per ogni parola
- **Attributi**: lettera minuscola iniziale, maiuscola iniziale per ogni successiva parola
- **Metodi**: lettera minuscola iniziale, maiuscola iniziale per ogni successiva parola

Progettazione OO: *esempi di CLASSI*

- Studenti di una scuola
- Dipendenti Banca
- Abitanti di una città
- Automobili
- Felini
- Mezzi di trasporto

UML : *Unified Modelling Language*

- Un **linguaggio di modellazione** permette di specificare, visualizzare e documentare un processo di sviluppo OO
- I **modelli** sono strumenti di comunicazione tra cliente e sviluppatori
- I linguaggi di modellazione più usati sono anche *standardizzati*



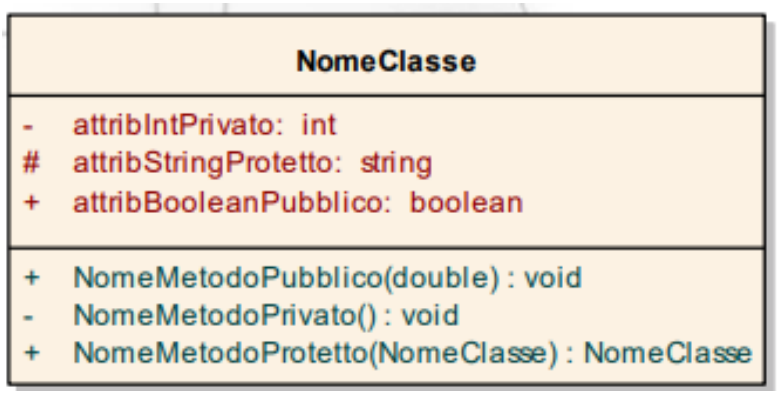
Progettazione OO : *Definizione Classe in UML*

Una **classe in UML** si rappresenta con un rettangolo avente tre compartimenti:

1. Nome della classe
2. Attributi
3. Metodi

Visibilità della classe:

- **+** Pubblica (il default per i metodi)
- **-** Privata (tipica per gli attributi)
- **#** Protetta (ereditarietà protetta, visibilità legata al package in Java)

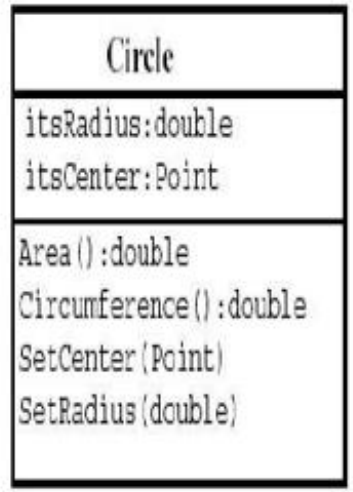


Classe espressa nella notazione standard **UML** (Unified Modeling Language)

Definizione Classe in UML

La classe :

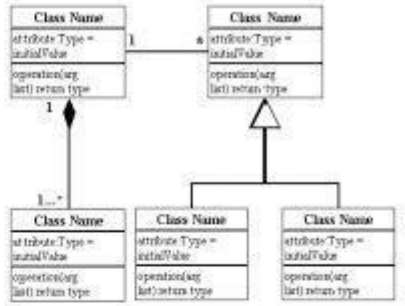
- Definisce
  - Uno stato persistente
  - Un comportamento
- La classe ha
  - Nome
  - Attributi
  - Operazioni
- Ha una rappresentazione grafica in forma di un rettangolo diviso in tre parti



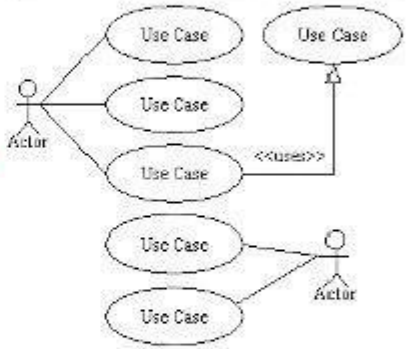
Una classe Circle,espressa nella notazione standard **UML** (Unified Modeling Language).

UML : *diagrammi principali*

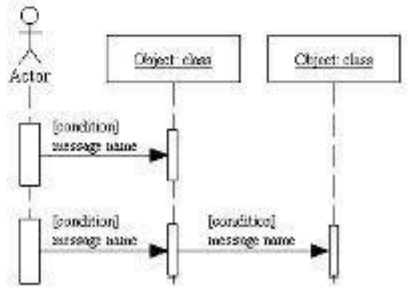
- Notazione per progetto OO
- Associata ad un metodo
- Parecchi tipi di diagrammi
  - Diagrammi di classe
    - Struttura statica
  - Diagrammi Casi d'Uso
    - Funzionalità
  - Diagrammi di sequenza
    - Vista temporale dello scambio di messaggi tra le classi



chi



Come



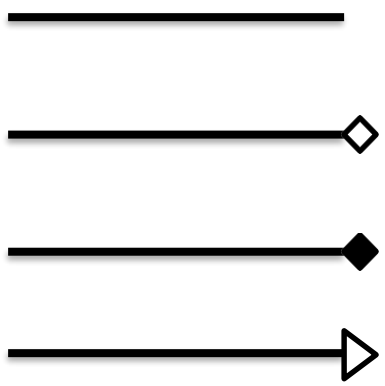
quando

Modello della CLASSE: *relazioni e utilità*

Una classe che non si interfacci con altre classi è sicuramente poco significativa in OOP. Abbiamo visto che gli oggetti, in un programma Object Oriented, interagiscono tra loro utilizzando lo scambio di messaggi per richiedere l'esecuzione di un particolare metodo.
Tale comunicazione consente di identificare all'interno del programma una serie di relazioni tra le classi in gioco la cui documentazione risulta essere assai utile in fase di disegno e di analisi.

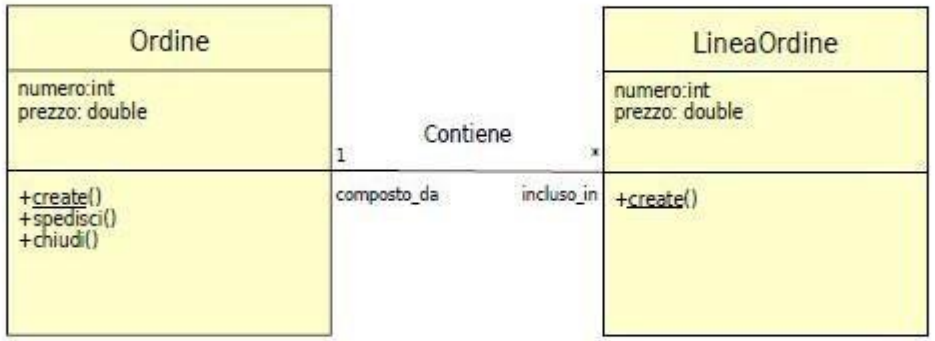
Le più comuni relazioni tra classi, in un programma ad Oggetti sono identificabili in tre tipologie:

- **Associazione** (relazione tra classi)
- **Aggregazione** (relazione poco forte)
- **Composizione** (relazione molto forte)
- **Ereditarietà** (relazione tra superclasse e classe figlia)



Modello della CLASSE: *associazione*

- Un'associazione rappresenta una connessione tra due o più classi
- La classe ha la responsabilità di notificare una certa informazione ad un'altra classe
- Bidirezionale
- La molteplicità indica quanti oggetti di una classe possono far riferimento ad ogni oggetto dell'altra



Modello della CLASSE: *associazione*

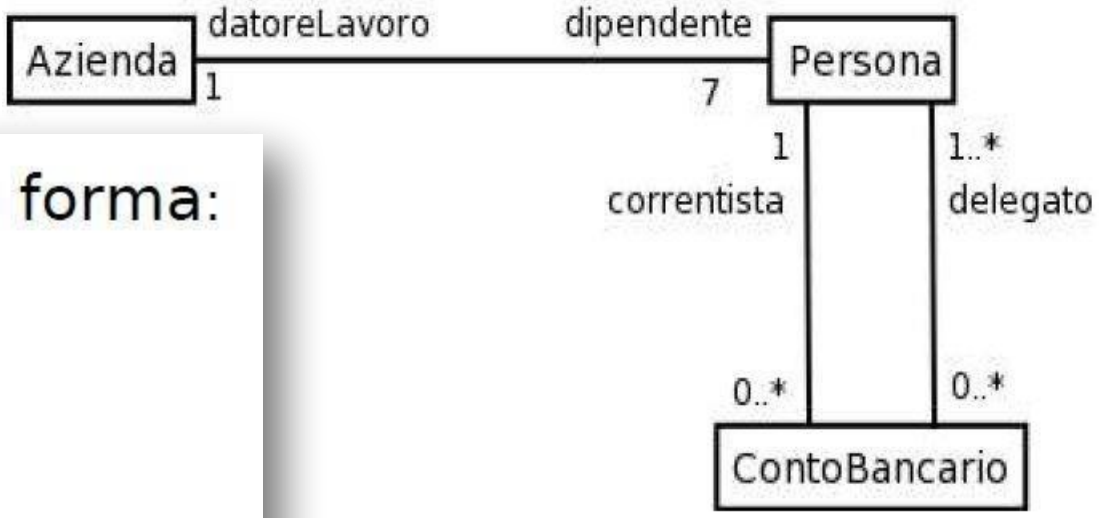
- Molteplicità:

Un'Azienda impiega molte Persone

Una Persona lavora per un'unica Azienda

Modello della CLASSE: *molteplicità*

Esempio



Le molteplicità sono della forma:

- 0..1 Zero o 1
- 1 Esattamente 1
- 0..* Zero o più
- * Zero o più
- 1..* 1 o più
- 1..6 da 1 a 6
- 1..3,7 da 1 a 3, *oppure* 7

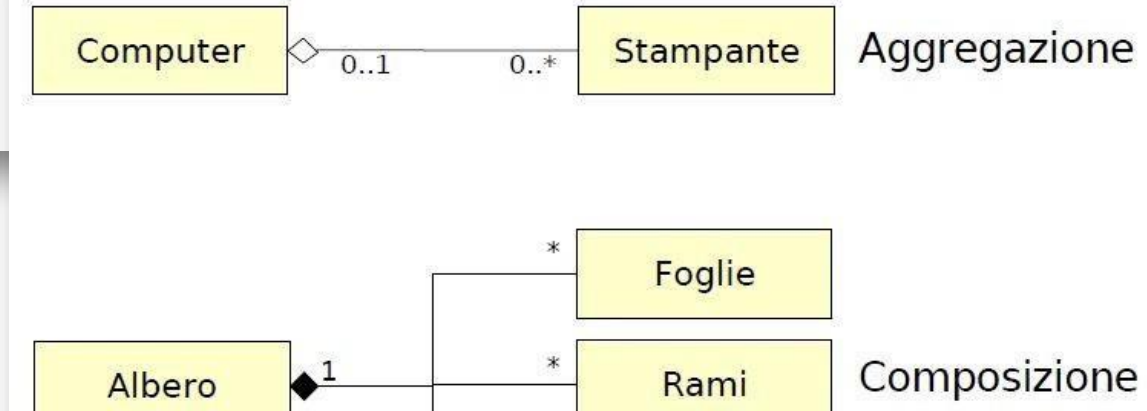
Modello della CLASSE: *aggregazione e composizione*

Aggregazione e *composizione* sono speciali forme di associazione che specificano una relazione *whole-part* *(tutto-parte)* tra l'aggregato e le parti componenti.

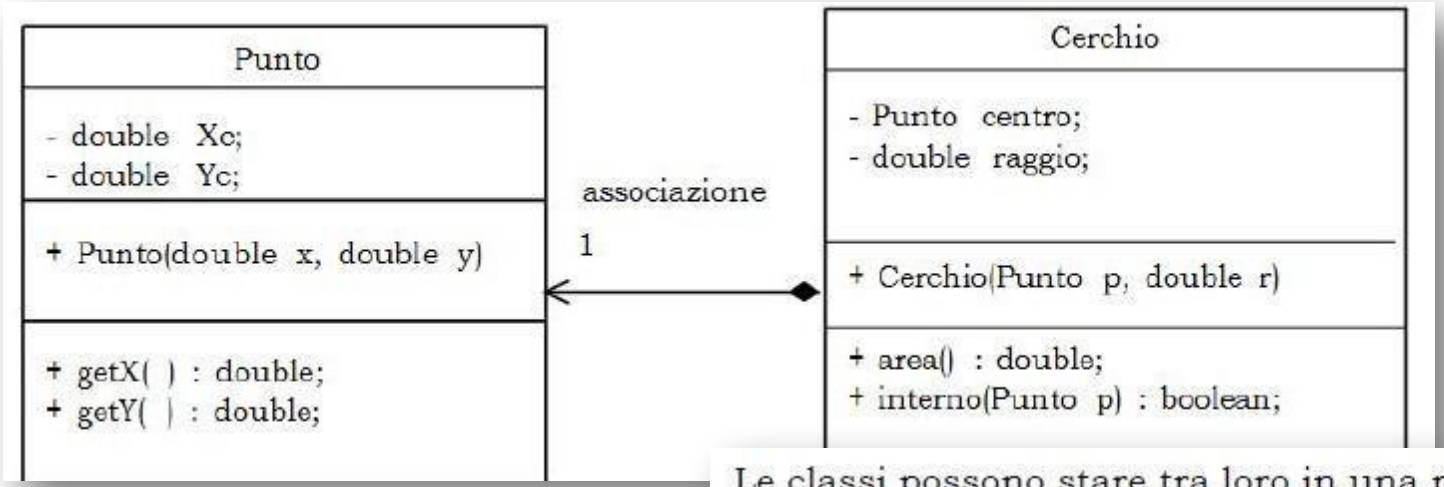
- Aggregazione: relazione poco forte (le parti esistono anche senza il tutto)
- Composizione: relazione molto forte (le parti dipendono dal tutto, per esempio i muri e la stanza)

Aggregazione e Composizione

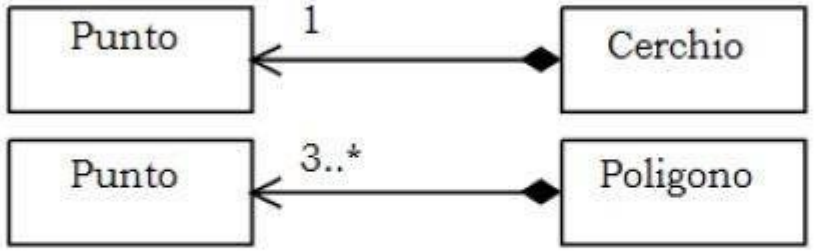
Una speciale forma di associazione che specifica una relazione tra la parte intera (*aggregato*) ed i suoi componenti (*parti*)



Modello della CLASSE : *Esempio di composizione*



Le classi possono stare tra loro in una relazione associativa di Composizione e questo si indica con la seguente simbologia



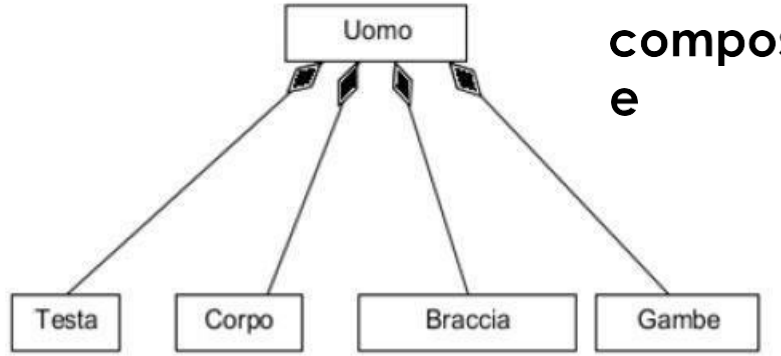
due classi sono in relazione di composizione se la prima di esse risponde alladomanda "HA UN" e la seconda di conseguenza si configura come la parte di un tutto. Nel primo caso disegnato si dice che la classe Cerchio ha un Punto (ilcentro) che le appartiene come sua parte. Nel secondo caso disegnato la relazione ci informa del fatto che un Poligono deve avere come sue parti 3 o più punti.

Modello della CLASSE : *Esempio di relazioni*

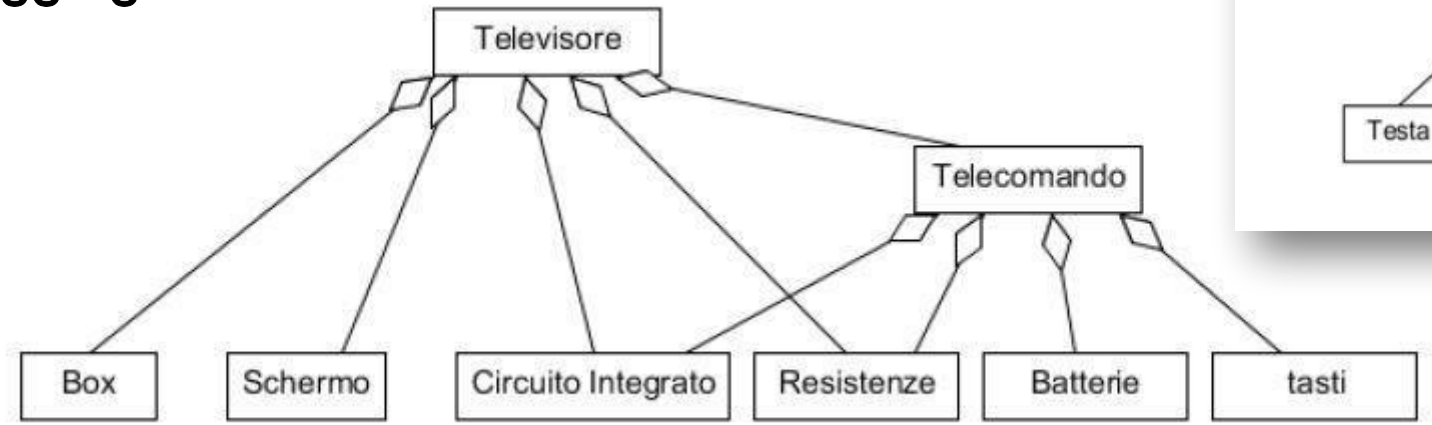
- Associazioni (*associations*) sono relazioni strutturali
- Generalizzazioni (*generalizations*) sono relazioni di ereditarietà.
- Composizione e Aggregazione (*composition and aggregation*) speciali relazioni strutturali

Figura 1. Diagramma di una relazione di associazione

composizione



aggregazione

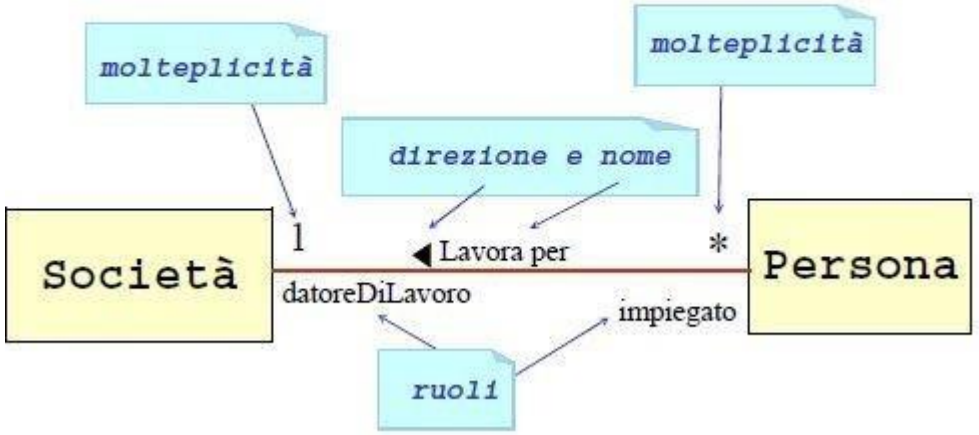


Modello della CLASSE : *Riepilogo*

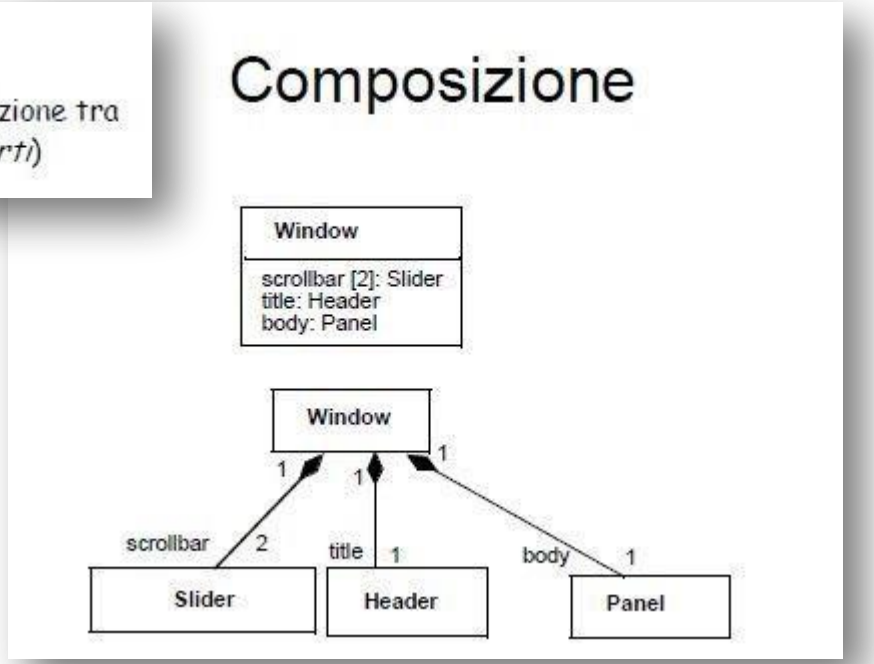
Aggregazione e Composizione

Una speciale forma di associazione che specifica una relazione tra la parte intera (*aggregato*) ed i suoi componenti (*parti*)

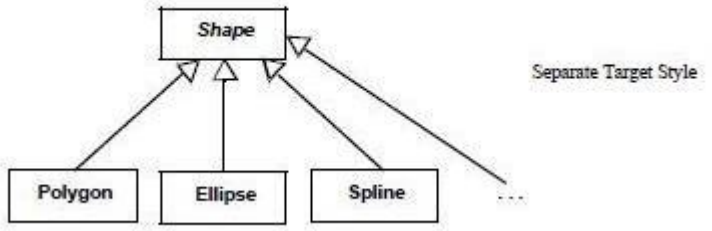
Associazioni



Composizione



Ereditarietà



Processor.java

```java
package com.journaldev.examples;

import java.util.Arrays;

public class Processor {

    public void process(int i, int j) {
        System.out.printf("Processing two integers:%d, %d", i, j);
    }

    public void process(int[] ints) {
        System.out.println("Adding integer array:" + Arrays.toString(ints));
    }

    public void process(Object[] objs) {
        System.out.println("Adding integer array:" + Arrays.toString(objs));
    }
}

class MathProcessor extends Processor {

    @Override
    public void process(int i, int j) {
        System.out.println("Sum of integers is " + (i + j));
    }

    @Override
    public void process(int[] ints) {
        int sum = 0;
        for (int i : ints) {
            sum += i;
        }
        System.out.println("Sum of integer array elements is " + sum);
    }

}
```

Overloading: Same Method Name but different parameters in the same class

Overriding: Same Method Signature in both superclass and child class