

# 01 – Il Paradigma Ad Oggetti

- ❑ Sistemi informativi
- ❑ Modellare la realtà (1):  
classi e oggetti
- ❑ Modellare la realtà (2):  
relazioni
- ❑ Progettazione ad oggetti
- ❑ Primo programma Eclipse/JUnit

# Sistemi informativi

- ❑ La realtà che ci circonda è estremamente complessa
  - Molteplicità di elementi, interazioni, comportamenti, ...
- ❑ Il compito del software è modellare porzioni di tale realtà allo scopo di realizzare sistemi informativi
  - In grado di raccogliere, elaborare, trasmettere dati per offrire servizi ai loro utilizzatori

# Il ruolo dei linguaggi di programmazione

- ❑ Il comportamento di un sistema informativo non dovrebbe dipendere dal modo con cui è stato realizzato
  - Ma solo dalle sue specifiche
- ❑ Alcune proprietà (manutenibilità, possibilità di riuso, ...) possono però dipendere molto dalle scelte fatte in fase di progetto
  - In particolare dalla metodologia di progetto e dai linguaggi di codifica adottati
- ❑ Alcuni linguaggi offrono astrazioni più potenti
  - L'esperienza mostra che sono più adatti a modellare situazioni complesse e in forte evoluzione

# Programmazione Strutturata

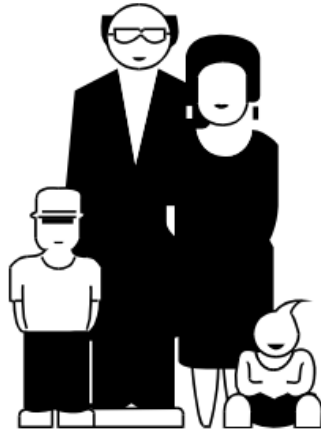
- ❑ Il codice è suddiviso in moduli
- ❑ Ogni modulo è composto da procedure e strutture dati
  - Le procedure operano sui dati, modificandoli
  - Riflette l'organizzazione di un elaboratore
  - Modello soggiacente al linguaggio "C"
- ❑ Non c'è un legame forte tra procedure e dati:
  - Qualsiasi procedura in grado di accedere ad un dato lo può modificare
  - Approccio che non scala al crescere delle dimensioni del problema

# Un'arte difficile

- ❑ La programmazione richiede molte abilità:
  - Creatività
  - Capacità di analizzare e risolvere problemi
  - Capacità di pensiero strutturato
  - Precisione e attenzione ai particolari
  - Conoscenza approfondita del linguaggio di programmazione adottato
  - ... e altre cose!
- ❑ L'approccio strutturato non semplifica il problema
  - Basso livello di astrazione
  - Il numero di dettagli di cui occuparsi cresce troppo al crescere della dimensione del problema trattato

# Ragionare ad Oggetti ?

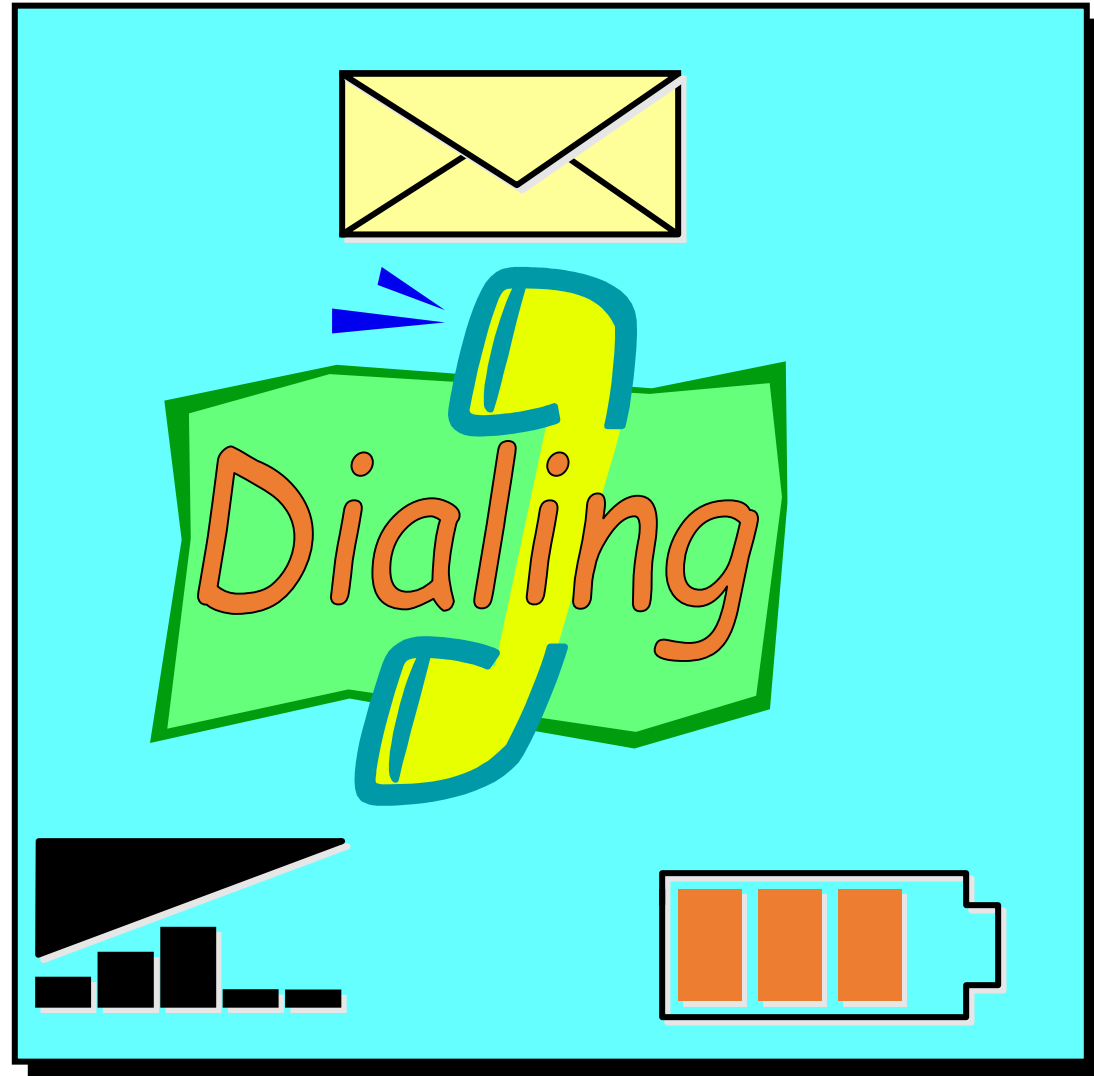
**OOP** è un metodo per costruire programmi in un modo che rispecchia l'organizzazione degli oggetti del mondo reale a differenza della programmazione **procedurale**



# Oggetti e realta

- ❑ Il mondo fisico è costituito da un insieme di oggetti variamente strutturati che interagiscono tra loro
- ❑ Ciascuno è dotato di:
  - Una propria **identità** (è riconoscibile)
  - Uno **stato** (ricorda la storia passata)
  - Un **comportamento** (reagisce a stimoli esterni in un modo prevedibile)
- ❑ Si può estendere la metafora al software
  - Ogni entità logica che deve essere manipolata può essere immaginata come un “oggetto”

# Stato





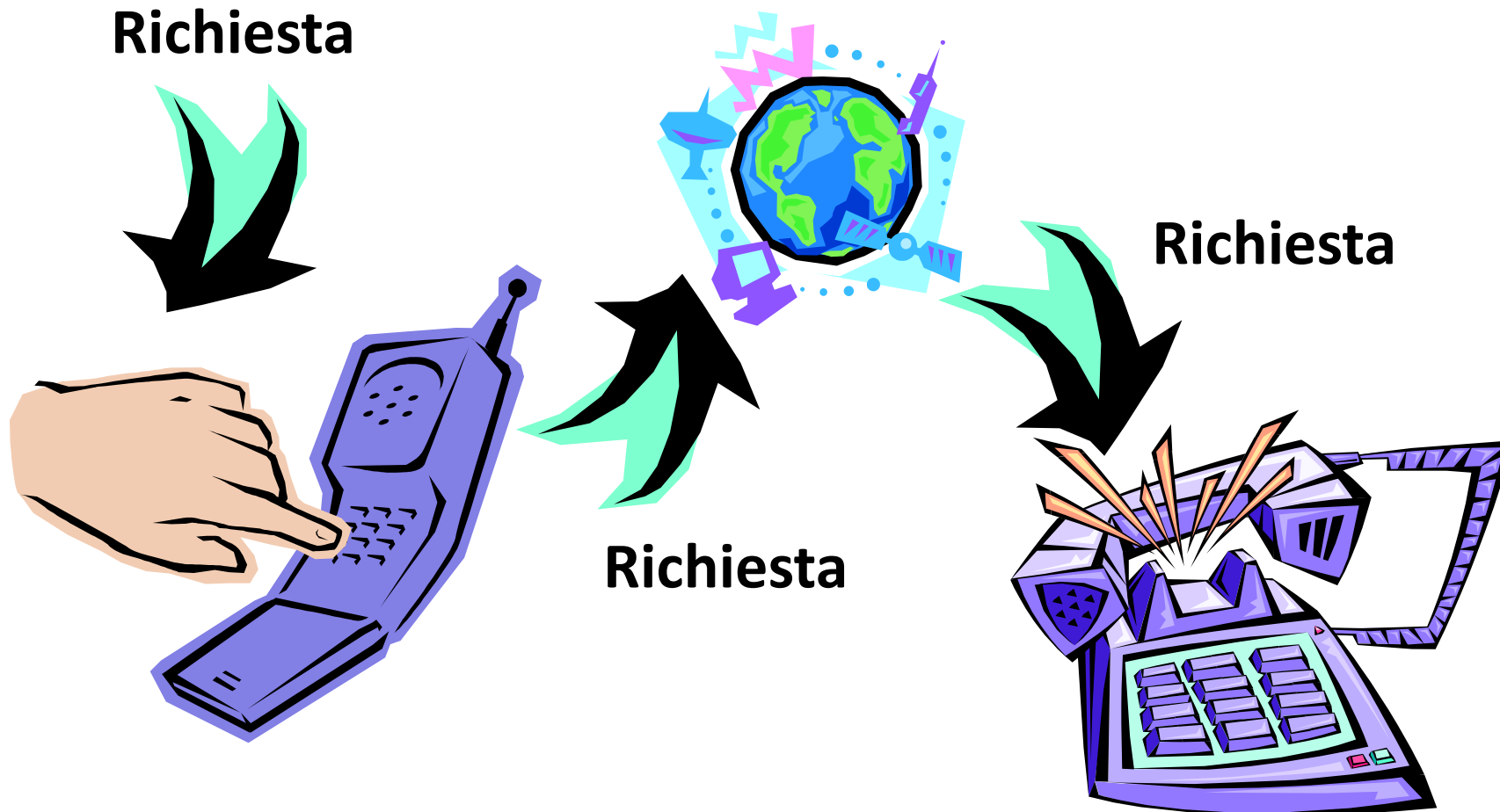
# Stato

- ❑ Ogni oggetto ha uno stato:
  - L'insieme dei parametri caratteristici che contraddistinguono un oggetto in un dato istante
  - Riflette la storia dell'oggetto
- ❑ Composto da un gruppo di "attributi"
  - Ogni attributo modella un particolare aspetto dello stato
  - Può essere un valore elementare o un altro oggetto...
- ❑ Implementato mediante un blocco di memoria
  - Contiene i valori degli attributi
- ❑ Principio fondamentale: **incapsulamento**
  - Lo stato "appartiene" all'oggetto
  - Un utente esterno non può manipolare direttamente lo stato

# Stato

- ❑ Gli **attributi** sono i dati che differenziano un oggetto da un altro
  - Un veicolo potrebbe avere i seguenti attributi
    - Stato: In movimento, fermo, in riparazione
    - Velocità: in Km/h
    - Proprietario: nome
- ❑ In una classe gli attributi sono definiti da variabili: posizioni nel computer dove si memorizzano informazioni. Ogni oggetto può avere valori differenti per le proprie variabili, chiamate **variabili di istanza**.
- ❑ Una variabile di istanza, detta anche **variabile oggetto**, definisce l'attributo di un particolare oggetto. La classe di un oggetto definisce il tipo di attributo e ogni istanza contiene il proprio valore per quell'attributo.
- ❑ Una variabile di classe (**static**) definisce un attributo comune, condiviso fra tutte le istanze (oggetti) della classe

# Comportamento



# Comportamento 1

- ❑ Gli oggetti interagiscono a seguito di “richieste” esterne (**messaggi**)
  - Dotate di eventuali parametri che ne specificano i dettagli
- ❑ Ogni oggetto sa reagire ad un ben determinato insieme di messaggi
  - Costituiscono la sua interfaccia
- ❑ Ad ogni richiesta è associato un comportamento
  - Modifica dello stato
  - Invio di richieste verso altri oggetti
  - Comunicazione di informazioni (risultato)
- ❑ Implementato attraverso un blocco di codice (**metodo**)
  - Contiene la sequenza delle operazioni da svolgere
- ❑ Principio fondamentale: **delega**
  - Chi effettua la richiesta non vuole conoscere i dettagli di

# Comportamento 2

- ❑ E' il modo in cui una classe di oggetti è in grado di svolgere operazioni su di essa o su altri oggetti
- ❑ Modifica attributi oggetto, ricezione e invio informazioni da altri oggetti
- ❑ Il comportamento di una classe si si ottiene utilizzando i metodi
- ❑ I metodi possono essere
  - Di **istanza**, comunemente chiamato metodi relativi al singolo oggetto
  - Di **classe** (static), relativi alla classe senza bisogno di istanziare oggetti

# Metodi e funzioni

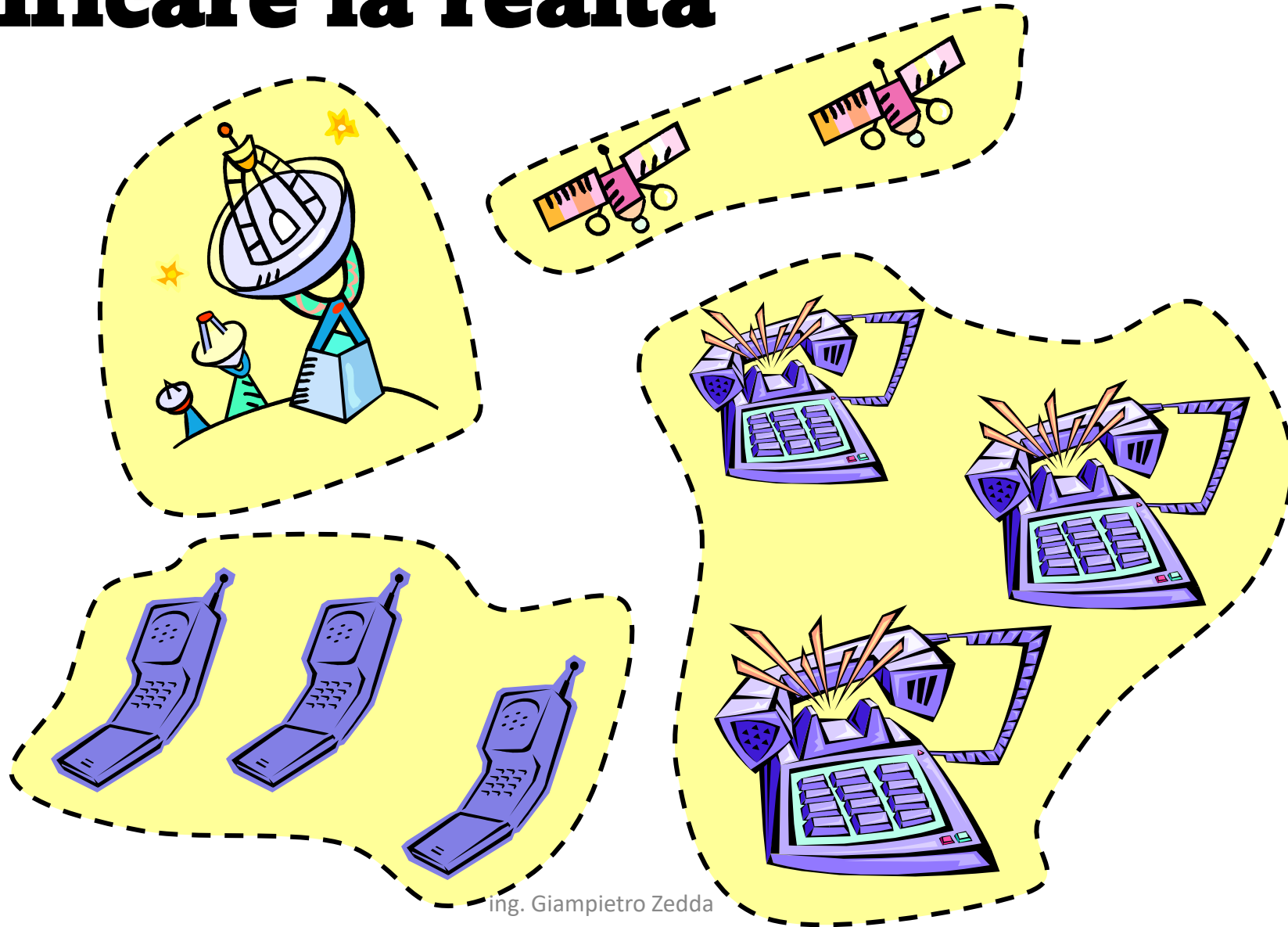
## □ Un metodo è simile ad una funzione

- Ha un nome
- Può avere dei parametri
- Può avere un valore di ritorno
- È composto da un insieme di istruzioni

## □ Ma...

- Agisce su un oggetto specifico (il destinatario del metodo)
- Il codice effettivamente eseguito dipende dal destinatario
  - cosa succede quando si compone un numero su un cellulare rispetto a quando si compone lo stesso numero su un telefono fisso?

# Classificare la realtà



# Classificare la realtà

- ❑ Nel mondo reale esistono molti oggetti
  - Per ogni “tipo” di oggetto le persone hanno sviluppato un apposito “concetto”
- ❑ Il concetto esiste nella mente delle singole persone e indica:
  - le caratteristiche ed il comportamento atteso di un oggetto
  - Le sue relazioni (a livello concettuale) con altri oggetti
- ❑ I singoli oggetti esistono nella realtà
  - Hanno una propria identità ed un proprio stato
  - Si comportano (dovrebbero comportarsi) coerentemente con il modello concettuale



# Classi e realta

- ❑ Nella Realtà, dato un insieme di ‘cose’, spesso astraiamo dalle loro caratteristiche per arrivare a definire un ‘concetto’ che le rappresenta tutte
  - ‘Mario Rossi’ e ‘Luisa Bianchi’ sono tutte **Persone**
  - Microsoft Word, SUN Star Office sono tutti **Word Processor**
- ❑ Una classe si compone di due distinti tipi di informazioni: gli attributi e la funzionalità
- ❑ Una classe è un modello per creare un **oggetto**
- ❑ La creazione di un oggetto a partire da una classe è chiamata **istanziamento** e gli oggetti creati sono detti **istanze**

# Classi e oggetti

- ❑ Nei sistemi informativi, ogni oggetto appartiene ad una classe
  - È istanza della classe
  - La classe costituisce il “progetto” dell’oggetto e ne specifica i metodi e gli attributi
- ❑ Oggetti appartenenti alla stessa classe hanno lo stesso comportamento e lo stesso insieme di attributi
  - Ma possono avere stati (valori) differenti

# Classi e oggetti 2

- ❑ Gli oggetti sono caratterizzati da
  - un insieme di dati (stato)
  - un insieme di istruzioni (comportamento)
- ❑ Gli oggetti si “costruiscono” a partire dalla classe
  - Costruire un oggetto significa allocare un blocco di memoria e registrare al suo interno lo stato
  - Nel “progetto” (classe) sono contenute apposite istruzioni circa la corretta creazione/configurazione degli oggetti “nuovi”

# Classe vs. Oggetto

- Il “concetto” unificante è la **classe**
  - Definisce l’insieme delle caratteristiche comuni a diverse “cose”
  - Caratteristiche “strutturali” (**attributi**) e “comportamentali” (**metodi/operazioni**)
  - Es.: PERSONA, WORD\_PROCESSOR
- La singola “cosa” è l’**oggetto**
  - Istanza di una classe
  - Es.: Mario Rossi, Microsoft Word

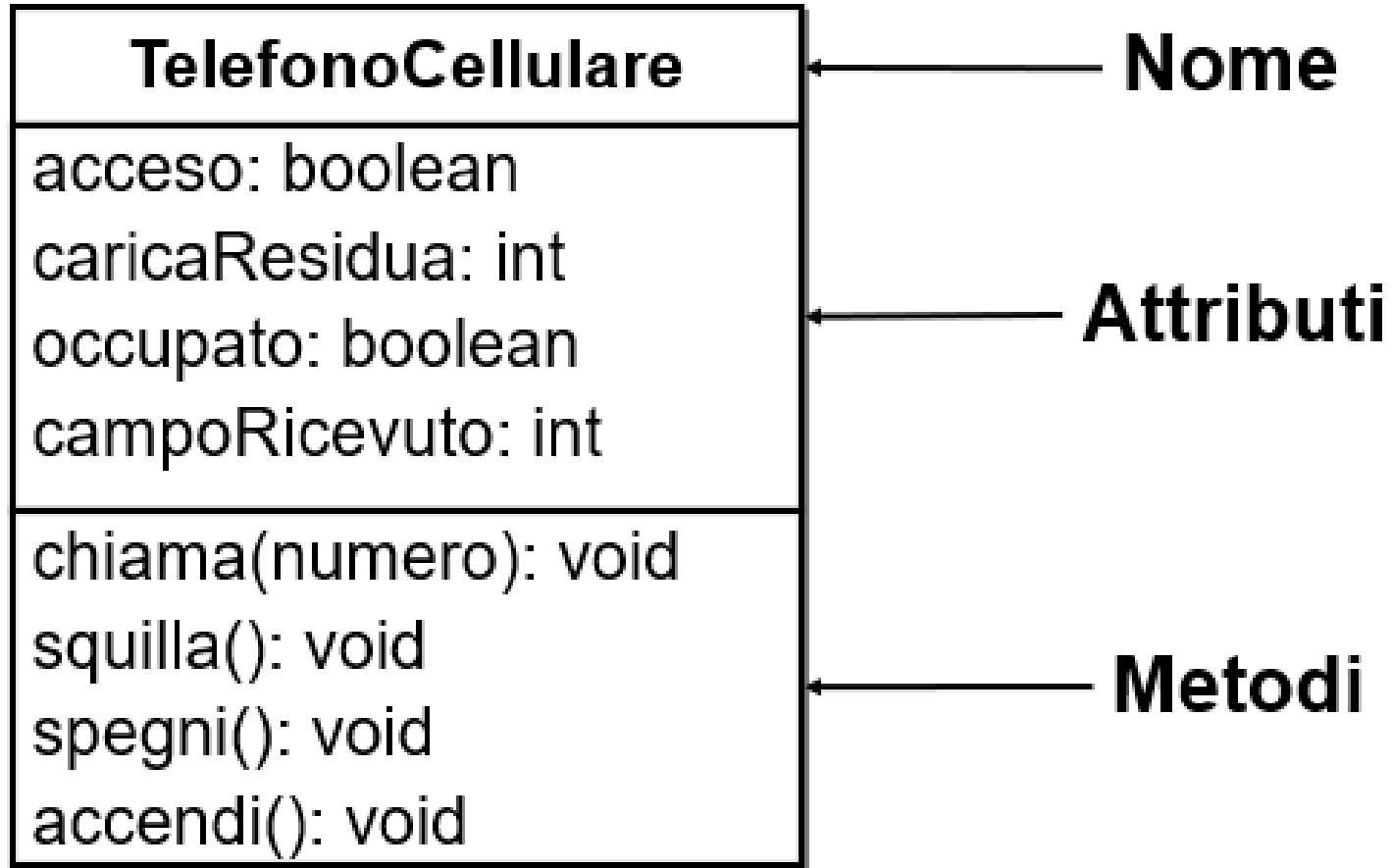
# Esempio

0



N	
	Telefono1 Numero: 011 5647044
	Telefono2 Numero: 011 5647091
N	Memoria

# Rappresentare le classi



Il compito del programmatore Java consiste nel creare una famiglia di Classi adeguata agli scopi del programma

# Esempi

## ☐ Sveglia

- Da quali attributi è composto il suo stato?
- Che tipo di informazione contengono?
- Quali metodi offre?
- Qual è la logica associata a ciascun metodo?



## ☐ Calcolatrice tascabile

- Si trascurino cifre decimali, memoria, percentuale
- Problema più complesso!





# Relazioni tra le classi

- ❑ Concetti (classi) diversi possono essere tra loro collegati:
  - Il motore è parte di un'automobile
  - Una persona è figlia di due genitori
  - Un veicolo è guidato da un autista
  - ...
- ❑ Si rappresentano tali legami attraverso linee che connettono le classi



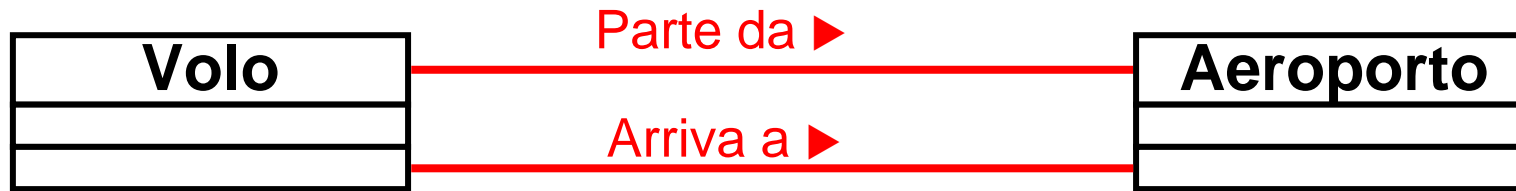
# Ruoli e molteplicità

- ❑ Ogni classe svolge un ruolo specifico in una relazione
  - Lo si esprime dando un nome alle estremità della linea
- ❑ Data una relazione che coinvolge un dato oggetto, l'altro ruolo può essere svolto da
  - 0..1 – zero o uno
  - $N$  – un numero specifico (ad esempio, 1, 2, ...)
  - \* – qualunque numero (scritto anche 0..\*)
  - 1..\* – uno o più oggetti

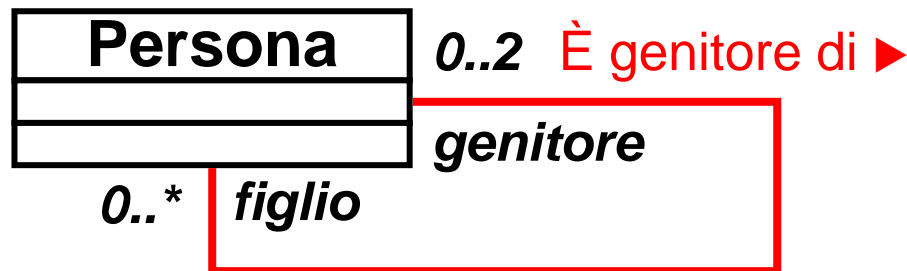


# Relazioni multiple

- ❑ Tra una data coppia di classi, possono esistere più relazioni che le uniscono




- ❑ Possono esistere relazioni che coinvolgono una sola classe

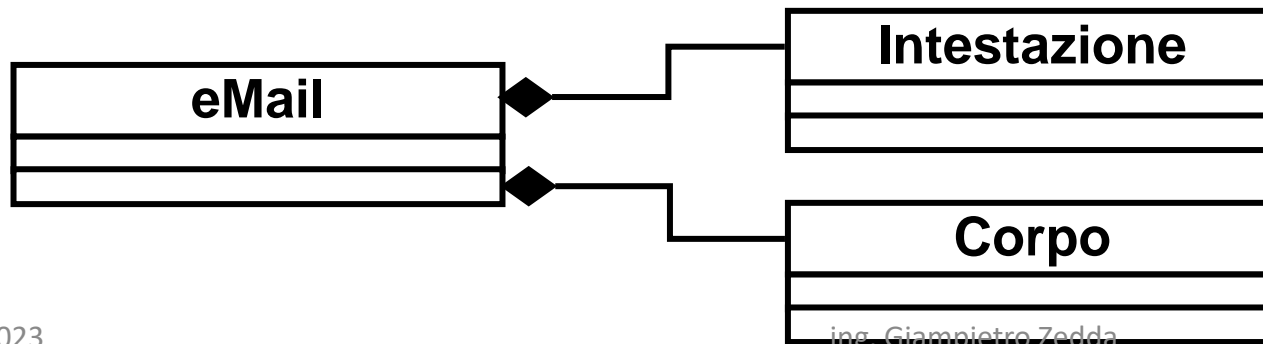


# Appartenenza


- ❑ Oggetti complessi sono composti da parti più semplici
  - Le funzionalità dell'oggetto composto sono il risultato della composizione delle funzionalità dei singoli elementi
  - Un'automobile “contiene” un motore, una carrozzeria, quattro ruote, ...
- ❑ Il ciclo di vita dell'oggetto composto può coincidere o meno con quello dei componenti:
  - Nel primo caso si parla di **composizione** (stretta)
  - Nel secondo, di **aggregazione** (debole)

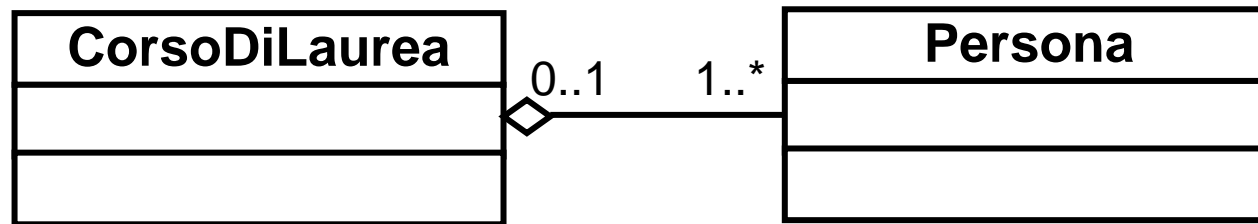
# Composizione (Appartenenza stretta)

- ❑ Le parti cessano di esistere quando l'oggetto composto cessa di esistere
  - La costruzione/distruzione dell'oggetto composto comporta la costruzione/distruzione dei componenti
- ❑ Un'entità può far parte di un solo composto
- ❑ Si indica con la notazione 

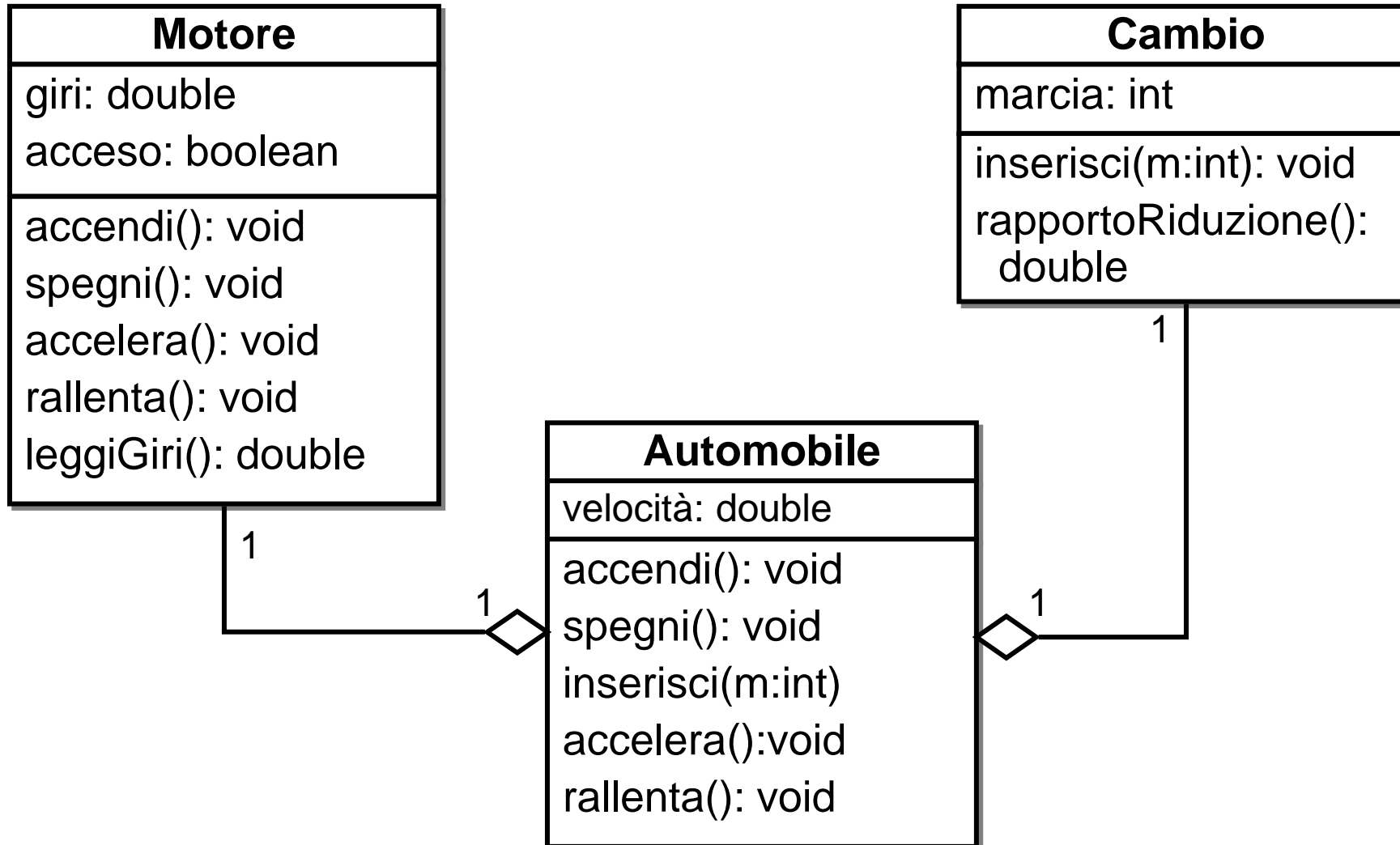


# Aggregazione (Appartenza debole)

- ❑ Le singole parti possono esistere indipendentemente dall'oggetto composto
- ❑ Una singola parte può essere aggregata a più oggetti composti
- ❑ Si indica con la notazione 



# Esempio



# Cosa esiste realmente?


- Nella realtà (i.e., nel “real world”, nell’applicazione) esistono solo gli oggetti
- Per rappresentare insiemi di oggetti (i.e., i concetti) utilizzo le classi
  - Non esistono le classi, esistono (i.e., “vivono” nel mondo, “girano” nell’applicazione) solo istanze di classi

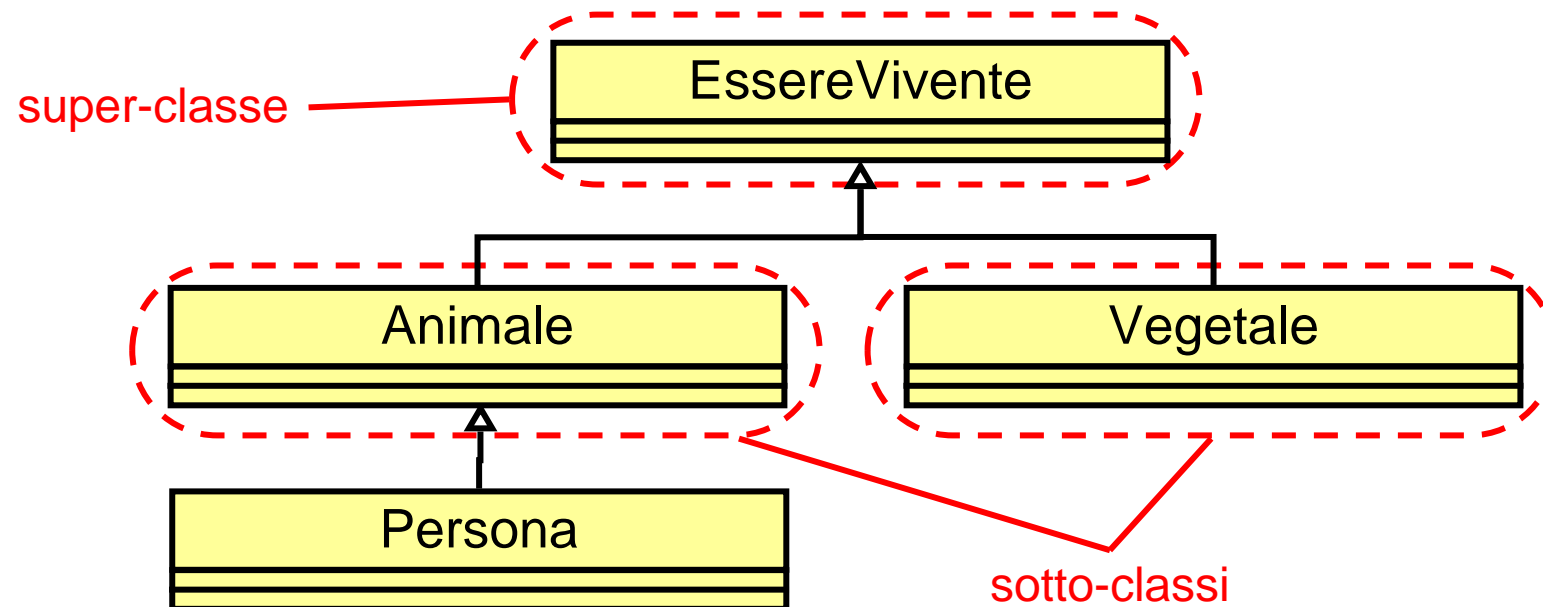


# Principi OOP incorporati in Java

- ❑ Organizzazione dei programmi in elementi chiamati **classi** e modalità con cui queste classi vengono utilizzate per creare gli **oggetti**
- ❑ Definizione di una classe tramite due aspetti della sua struttura: **comportamento** (metodi di funzionamento) e **attributi**
- ❑ Connessione delle classi tra loro in modo che una classe erediti le funzionalità di un'altra
- ❑ Collegamento delle classi fra di loro attraverso **package** e **interfacce**

# Gerarchie concettuali (1)

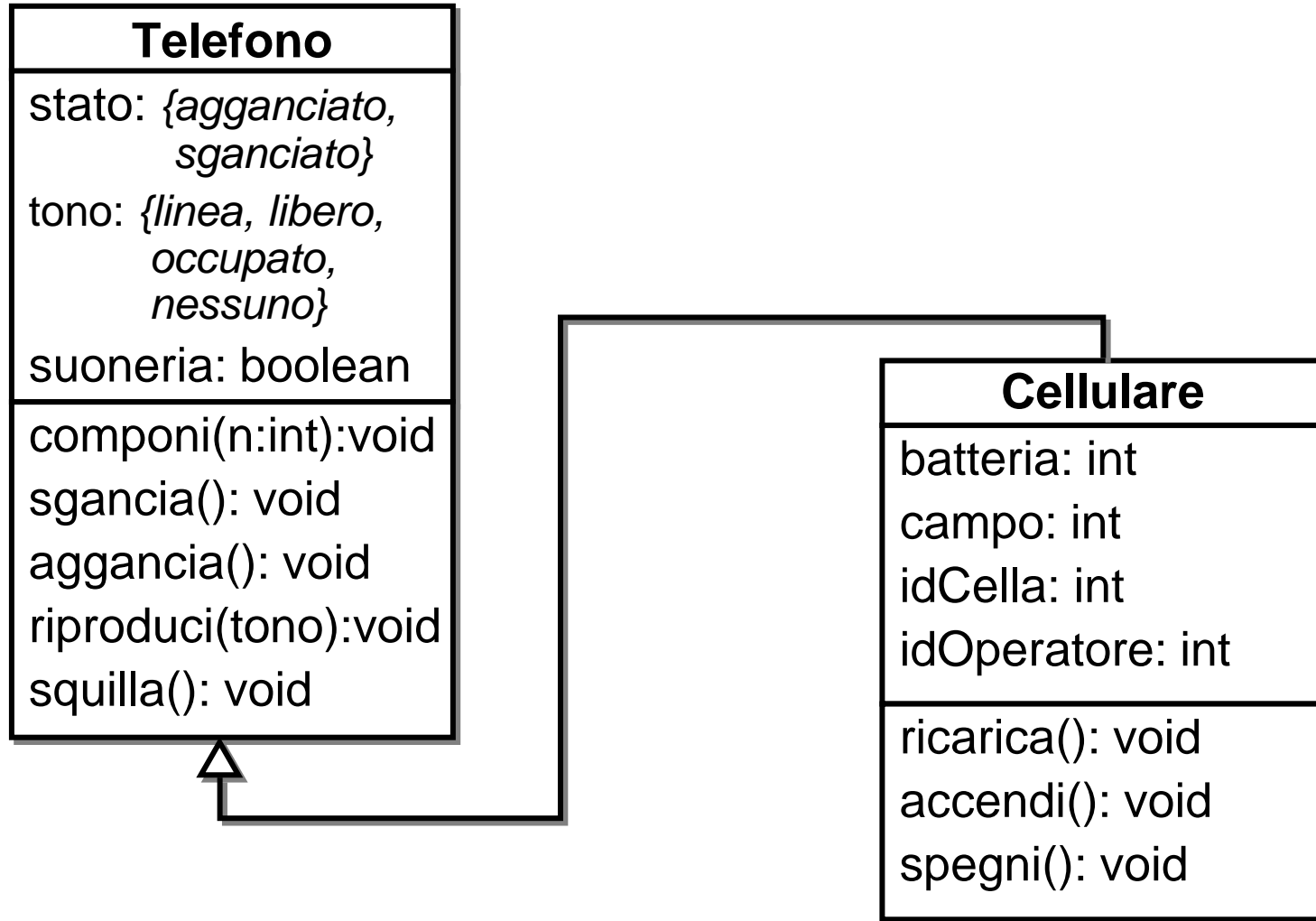
- ❑ Alcune classi modellano concetti generali che comprendono insiemi di concetti più specifici
  - Le caratteristiche (stato e comportamento) dei concetti più generici sono condivise dai concetti più specifici
- ❑ Si utilizza la notazione 



# Gerarchie concettuali (2)

- ❑ La relazione “è più specifico di” riveste un ruolo particolare nei linguaggi ad oggetti:
  - Fondamento dell'**ereditarietà**
- ❑ La classe più specifica “eredita” attributi, metodi, aggregazioni e altre relazioni da quella più generica
  - A questi, si affiancano attributi , metodi, aggregazioni e relazioni propri della sotto-classe
- ❑ La sotto-classe può ridefinire il comportamento legato ai metodi ereditati
  - A condizione di mantenerne intatta la definizione (numero e tipo dei parametri, tipo ritornato): **polimorfismo**

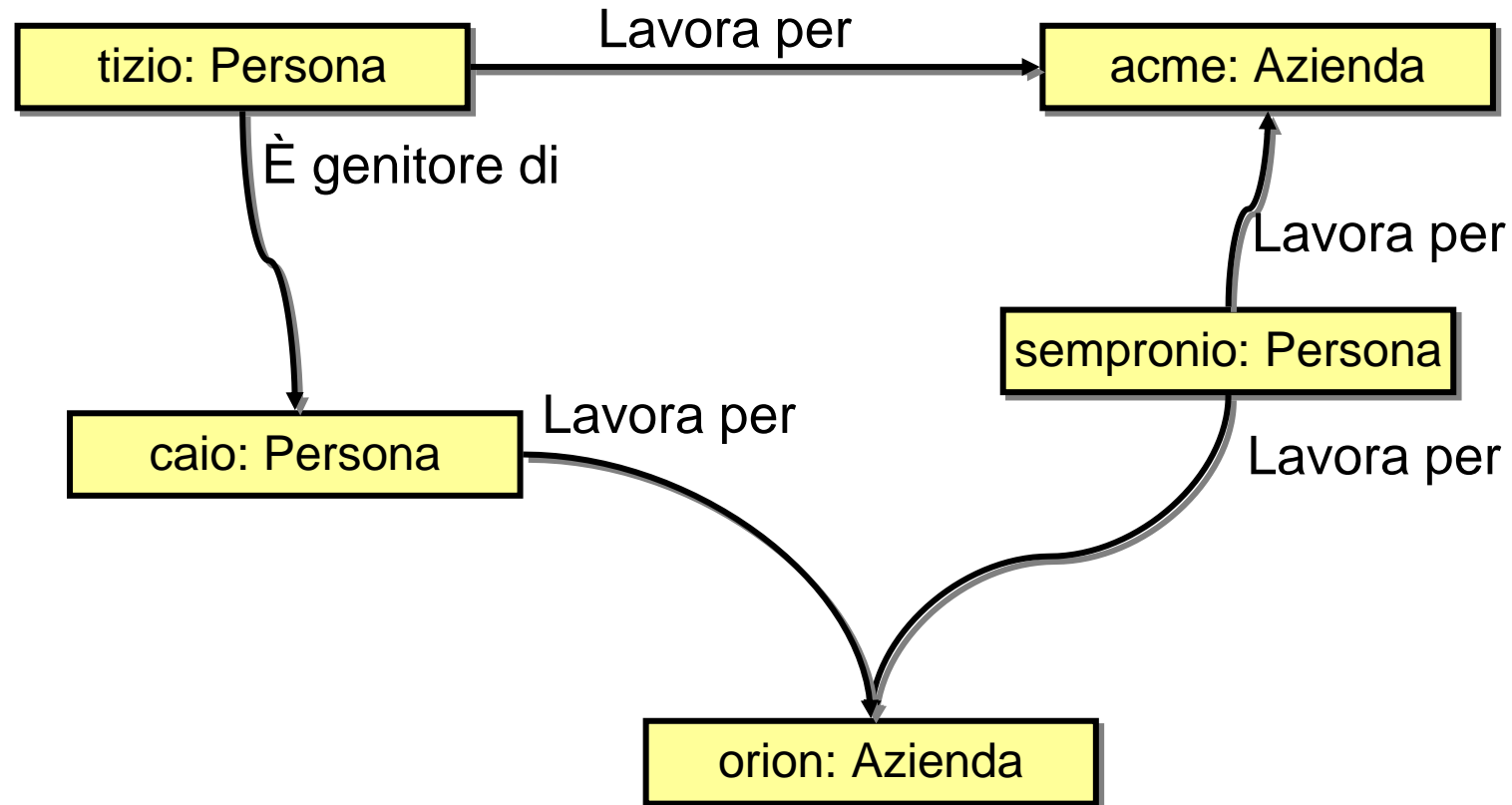
# Esempio



# Rappresentare gli oggetti

- ❑ Il mondo reale è costituito da oggetti
  - Così i programmi scritti con il paradigma ad oggetti sono composti da blocchi di memoria nel calcolatore
- ❑ Data una classe, possono esistere molte istanze
  - Ogni istanza è distinguibile dalle altre
- ❑ Si rappresentano gli oggetti attraverso i diagrammi delle istanze
- ❑ Per ogni oggetto, indicano:
  - il nome (arbitrario ma univoco)
  - la classe di appartenenza
  - i collegamenti cui partecipano (che devono soddisfare i vincoli sulla molteplicità)

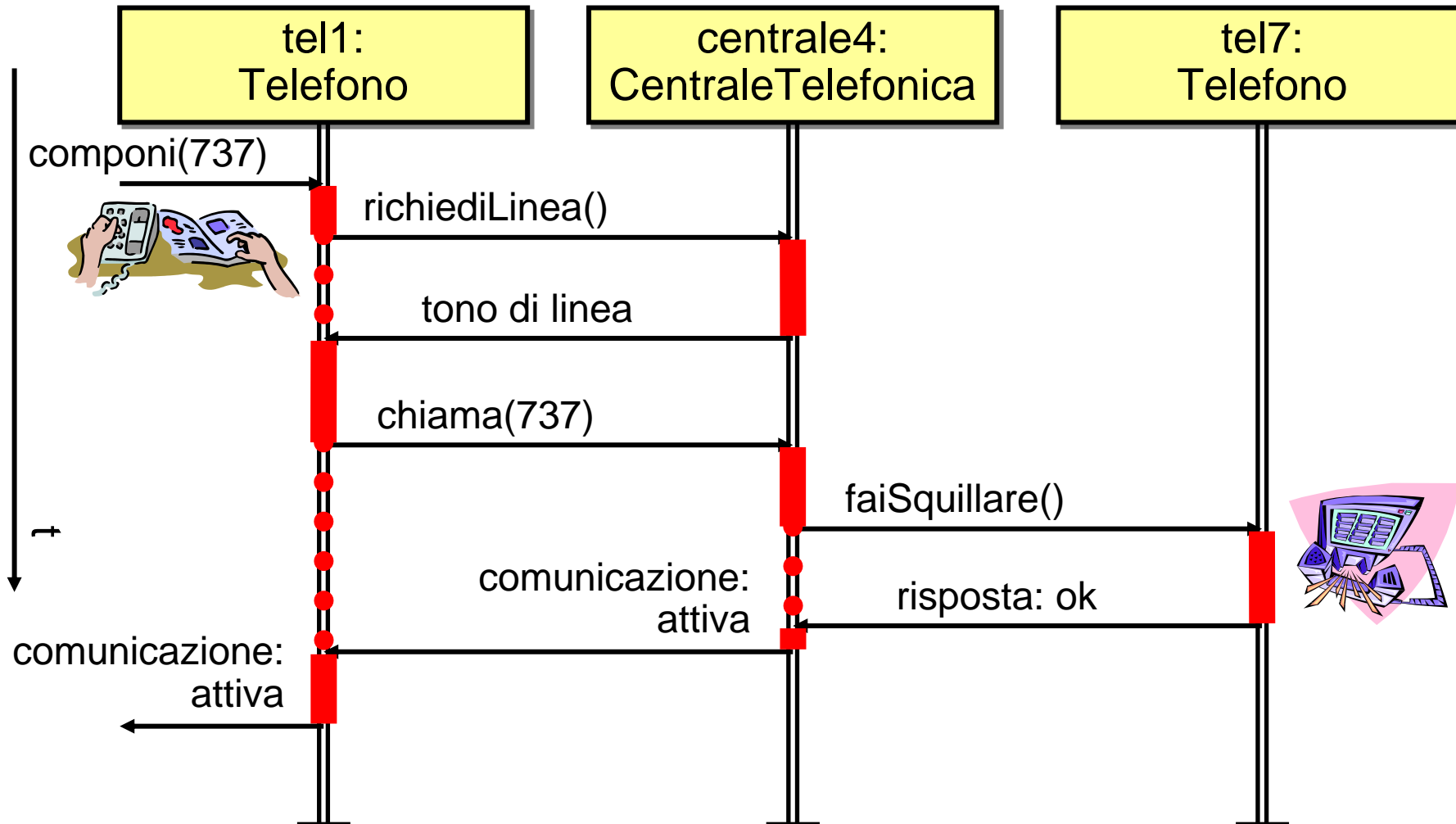
# Diagrammi delle istanze



# Interazione tra oggetti

- ❑ Un oggetto reagisce a richieste del mondo esterno
  - Eseguendo il codice associato al metodo richiesto
- ❑ La richiesta può comportare l'interazione con altri oggetti
  - La composizione di un numero su un telefono, richiede alla centrale telefonica di instaurare un collegamento con il destinatario
- ❑ Si può interagire solo con chi si conosce
  - Gli oggetti interagenti devono essere connessi da una qualche relazione
- ❑ Si rappresenta questo insieme di interazioni tramite i **“diagrammi di sequenza”**
  - Mostrano quali interazioni avvengono tra oggetti e la relativa sequenza temporale

# Diagrammi di sequenza





# Il processo di creazione del software (1)

- Dapprima, occorre capire **cosa** viene chiesto di fare
  - **Analisi**: definisce i termini del problema
  - Traduce le parole (ambigue) del committente in requisiti **verificabili**
- Poi, occorre capire **come** farlo
  - **Progetto**: definisce l'architettura della soluzione
  - Sceglie come realizzare i requisiti attraverso classi, oggetti, relazioni, ...

# Il processo di creazione del software (2)

- ❑ Quindi è utile definire come **verificare**
  - Progetto del **test**: definisce i controlli da attuare
  - Traduce i requisiti in un insieme di misure
- ❑ Da ultimo, si **scrive** e verifica il codice
  - **Sviluppo**: produce la soluzione
  - Implementa l'architettura
- ❑ Il processo è iterativo
  - Le diverse fasi si susseguono **raffinando progressivamente** la soluzione

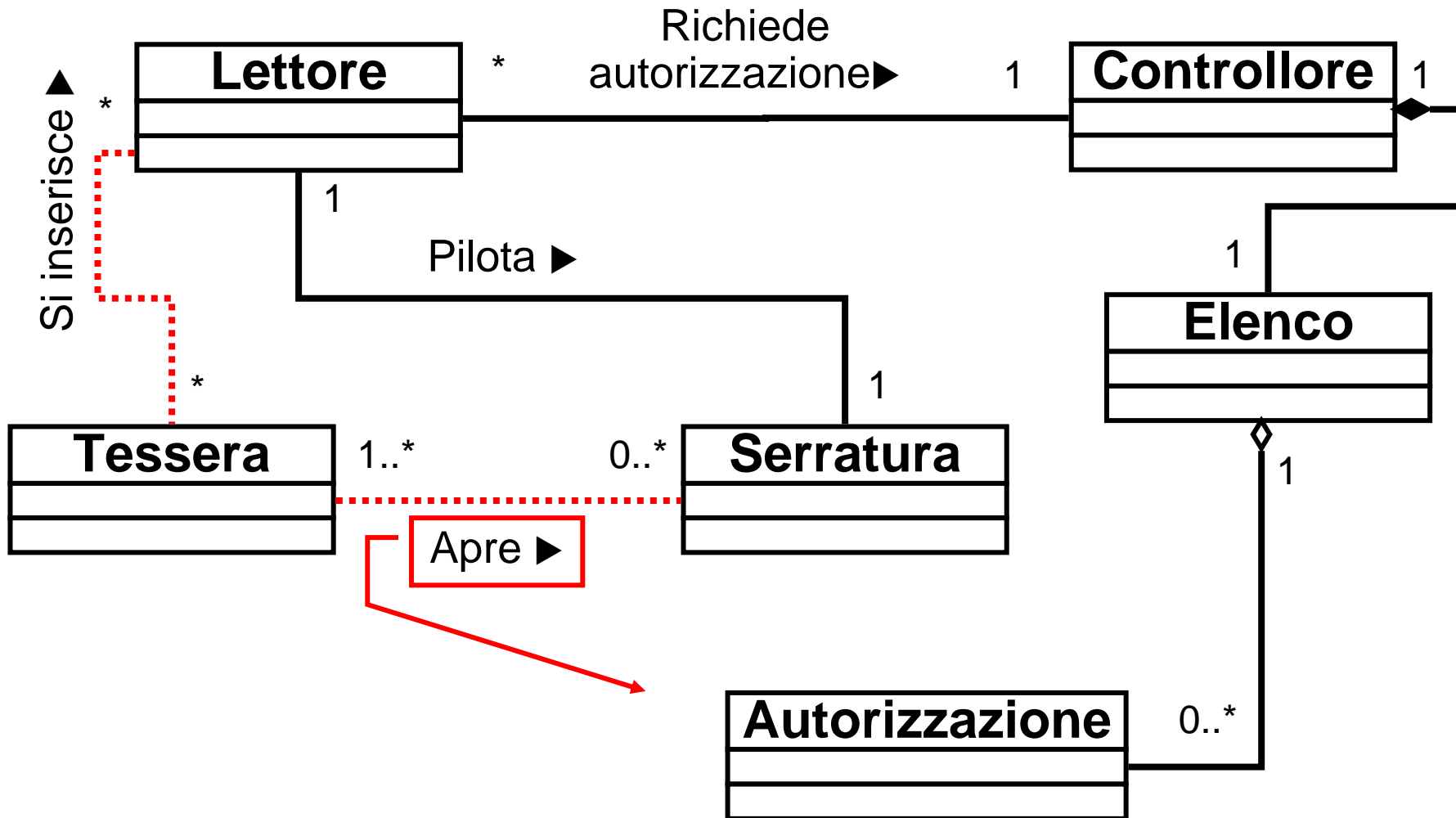
# Progettare ad oggetti

- ❑ Si parte dal testo delle specifiche
  - Si individuano i nomi e i verbi
- ❑ Tra i nomi, si individuano le possibili classi di oggetti
  - Con i relativi attributi
- ❑ Tra i verbi, si individuano metodi e relazioni
  - Di solito, i verbi di azione si modellano come metodi ( “X apre Y” ), quelli di stato come relazioni ( “A si trova presso B” )
  - L’interazione tra due oggetti sottende l’esistenza di una relazione tra gli stessi
  - Attenzione alle forme passive e ai sostantivi deverbali!

# Implementare le relazioni

- ❑ Se esiste una relazione dall'oggetto "A" verso l'oggetto "B", occorre memorizzare da qualche parte questa informazione
  - Il modo di farlo dipende dalla molteplicità della stessa
- ❑ Relazioni di tipo  $n:1$  ( $n \geq 0$ ), possono essere implementate sotto forma di attributo nella classe di partenza
  - L'attributo ha come tipo la classe destinazione
  - Il valore dell'attributo è il riferimento all'oggetto destinazione
- ❑ Relazioni di tipo  $n:m$  ( $n \geq 0$ ,  $m > 1$ ) richiedono l'utilizzo di strutture più complesse
  - Array di riferimenti e/o classi ausiliarie

# Controllo accessi: analisi



# Controllo accessi: analisi

<b>Tessera</b>
id: int
leggild(): int

<b>Serratura</b>
apri(): void

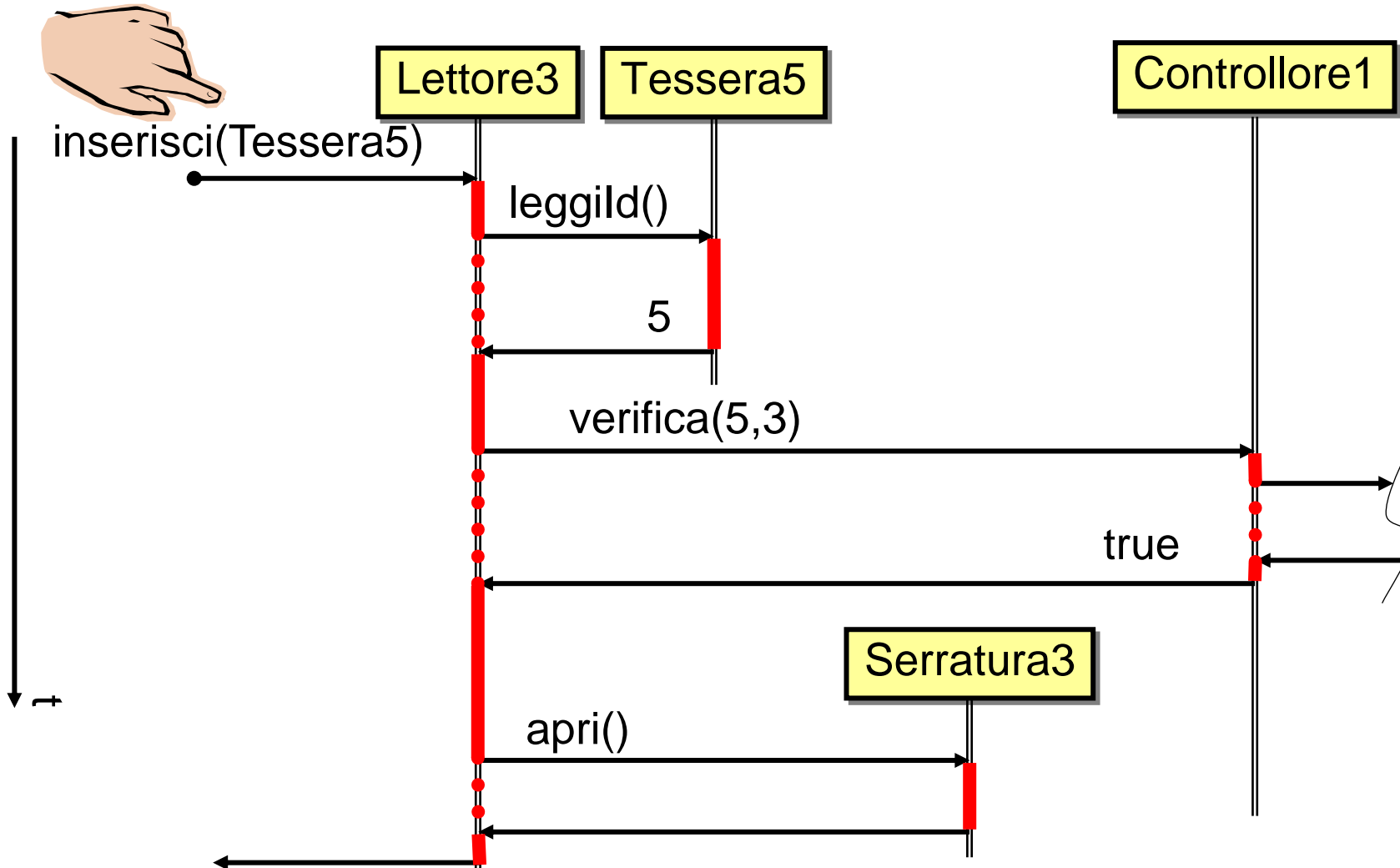
<b>Elenco</b>
numAut: int
cerca( idTessera:int, idLettore:int ) : boolean

<b>Lettore</b>
id: int
valida(Tessera): void

<b>Controllore</b>
verifica( idTessera: int, idLettore: int ) : boolean

<b>Autorizzazione</b>
idTessera: int idLettore: int
confronta( tessera:int, lettore:int): boolean

# Controllo accessi: diagramma di sequenza



# Nomenclatura

- A seconda degli autori (e della fase del ciclo di vita in cui li utilizzo, i.e., analisi concettuale vs. progettazione di dettaglio), i seguenti termini vengono usati come sinonimi, o con significati molto simili
  - attributo  $\approx$  campo  $\approx$  proprietà
  - metodo  $\approx$  operazione  $\approx$  servizio



# Nomenclatura

- A seconda degli autori (e della fase del ciclo di vita in cui si trova il modello concettuale vs. quello di dettaglio), i seguenti termini sono usati come sinonimi o termini molto simili

- attributo  $\approx$  campo  $\approx$  proprietà
- metodo  $\approx$  operazione

*(a livello intuitivo e poco rigoroso) Gli attributi sono delle variabili che ogni oggetto ha; (in modo più rigoroso) sono la definizione dei valori caratteristici di ogni singola istanza*

*Un metodo è una funzione che tutti gli oggetti possono svolgere, personalizzando il comportamento attraverso l'utilizzo degli attributi ( $\approx$  parametri impliciti della funzione)*

# Istanziamento

- In OO+ esiste l'operatore `new()` che crea un nuovo oggetto istanza di una classe `C`

```
class PERSONA {  
    string nome;  
    string cognome;  
    date data_nascita;  
}
```

Definizione di una  
classe

Dichiarazione di una  
**variabile oggetto**

```
PERSONA o = new PERSONA();
```

Istanziamento  
di un **oggetto**

# Riassumendo ...

- ❑ Gli oggetti sono caratterizzati da
  - un insieme di dati (stato)
  - un insieme di istruzioni (comportamento)
- ❑ Gli oggetti si “costruiscono” a partire dalla classe
  - Costruire un oggetto significa allocare un blocco di memoria e registrare al suo interno lo stato
  - Nel “progetto” (classe) sono contenute apposite istruzioni circa la corretta creazione/configurazione degli oggetti “nuovi”

<h2>Concetti Fondamentali</h2> <p><b>Il mondo è composto da OGGETTI</b></p> <p>Un oggetto ha</p> <ul style="list-style-type: none"> <li>• un nome</li> <li>• uno stato</li> <li>• un comportamento</li> </ul> <p>La classe è il modello (o progetto) di un oggetto</p> <p>Gli oggetti si costruiscono a partire dalla classe</p>	<h2>Concetti Subordinati</h2> <p>Un oggetto viene rappresentato da</p> <ul style="list-style-type: none"> <li>• un blocco di memoria che ne conserva lo stato (attributi)</li> <li>• un blocco di codice che ne precisa il comportamento (metodi)</li> </ul> <p>La classe definisce:</p> <ul style="list-style-type: none"> <li>• l'insieme degli attributi</li> <li>• l'insieme dei metodi</li> <li>• come viene costruito lo stato iniziale degli oggetti che modella</li> </ul>	<h2>Ulteriori Dettagli</h2> <p>Un metodo assomiglia ad una funzione ma è legato ad un oggetto specifico</p> <p>Il blocco di codice relativo ai metodi è condiviso da tutti gli oggetti appartenenti alla stessa classe</p>
<p><b>Gli oggetti possono interagire tra loro</b></p> <p>A livello concettuale, le classi di due oggetti che interagiscono sono tra loro in relazione</p>	<p>Se due oggetti interagiscono, deve esistere tra loro una relazione</p> <p>Una relazione tra due classi indica la potenzialità dell'interazione tra gli oggetti istanza di tali classi</p>	<p>Una relazione tra classi si traduce in un (insieme di) attributo ulteriore che indica se e con chi uno specifico oggetto è collegato</p>
<p><b>Alcune relazioni hanno un significato particolare</b></p> <ul style="list-style-type: none"> <li>• "è più specifico di"</li> <li>• "contiene"</li> </ul>	<p>Una sottoclasse eredita attributi e metodi della classe più generica</p> <p>Un oggetto composto è formato da un insieme di oggetti componenti</p>	<p>Una sottoclasse può ridefinire il comportamento dei metodi ereditati (polimorfismo)</p> <p>L'appartenenza può essere "debole" (aggregazione) o "forte" (composizione)</p>

<h2>Concetti Fondamentali</h2> <p><b>Si descrive uno scenario ad oggetti attraverso diagrammi</b></p> <p>Il diagramma delle classi indica classi, attributi, metodi e relazioni</p> <p>Il diagramma delle istanze presenta un caso concreto, a livello di esempio</p> <p>Il diagramma di sequenza mostra come determinate istanze dialogano tra loro</p>	<h2>Concetti Subordinati</h2> <p>Per ogni relazione viene indicato</p> <ul style="list-style-type: none"> <li>• il nome</li> <li>• i ruoli</li> <li>• la molteplicità</li> </ul>	<h2>Ulteriori Dettagli</h2>
<p><b>Il software si progetta e realizza in un processo iterativo</b></p> <p>Successione di 4 fasi:</p> <ul style="list-style-type: none"> <li>• Analisi</li> <li>• Progetto</li> <li>• Definizione dei test</li> <li>• Sviluppo del codice</li> </ul> <p>Un metodo per individuare classi e relazioni si basa sull'analisi del testo</p>	<p>Inizialmente si considerano solo i requisiti fondamentali, ad ogni iterazione successiva si aggiungono dettagli e funzionalità</p> <p>Si applicano i test al codice sviluppato per validarlo</p> <p>I nomi tendono a corrispondere a classi ed attributi, i verbi a metodi e relazioni</p>	<p>Occorre implementare le relazioni:</p> <ul style="list-style-type: none"> <li>• quelle con cardinalità n:1 (esclusa l'ereditarietà) diventano attributi</li> <li>• quelle n:m utilizzano strutture complesse (array, classi ausiliarie)</li> </ul>

# Primo Programma Java classe Veicolo

Package Explorer JUnit x

Finished after 0,141 seconds

Runs: 1/1 Errors: 0 Failures: 0

> VeicoloTest [Runner: JUnit 5] (0,000 s)

```
1 package Arg03;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 import org.junit.jupiter.api.Test;
5
6 class VeicoloTest {
7
8     @Test
9     void test() {
10         Veicolo veicolo = new Veicolo();
11         veicolo.proprietario = "Rossi";
12         veicolo.targa = "T0362370";
13         veicolo.velocitaMax = 210;
14         veicolo.mostraAttributi();
15     }
16 }
17
18 }
```

ReadFileLin... Veicolo.java VeicoloTest... »15

Servers Console x Debug

<terminated> VeicoloTest [JUnit] C:\Progra  
Proprietario: Rossi  
Targa: T0362370  
VelocitaMax: 210

Veicolo.java x TestArraysS... ReadFileBuff... ReadFileLin...

```
1 package Arg03;
2 public class Veicolo {
3     String proprietario;
4     String targa;
5     int velocitaMax;
6
7     public Veicolo() {
8     }
9
10    void mostraAttributi() {
11        System.out.println("Proprietario: " + proprietario);
12        System.out.println("Targa: " + targa);
13        System.out.println("VelocitaMax: " + velocitaMax);
14    }
15 }
16 }
```