Programmazione ad oggetti

dI Roberta Molinari

Risoluzione di un problema

Prevede 4 passaggi

1. Analisi dei dati
2. Stesura dell'algoritmo
3. Codifica del programma
4. Test del programma

- La **programmazione imperativa procedurale** si concentra sulla strutturazione dell'algoritmo
- La **programmazione ad oggetti** si concentra sull'analisi dei dati

2

Programmazione procedurale

- Nella programmazione imperativa procedurale si utilizza la tecnica di **top-down**: si parte da un problema e lo si scompone in sottoproblemi.
- Il programma prevede la chiamata ad un main iniziale che richiama delle funzioni
- Il programma è suddiviso in moduli indipendenti

Svantaggi:

- Difficile descrivere sistemi complessi
- Difficile riutilizzare codice già scritto

OOP-Object Oriented Programming
Introduzione

- Nel mondo reale esistono oggetti semplici, autonomi ciascuno con le sue caratteristiche e funzionalità specifiche, in grado di fare o su cui è possibile fare alcune operazioni. Gli oggetti semplici possono essere uniti per formare oggetti più complessi

Es. un PC è formato da CPU, RAM, Scheda madre,… Ogni componente è autonoma e può essere creata da produttori diversi; non è necessario conoscerne la struttura interna, ma solo le caratteristiche, la funzionalità e le specifiche di interfaccia per creare un PC funzionante

OOP-Object Oriented Programming
Introduzione

- La **programmazione orientata agli oggetti OOP** si sviluppa intorno al 1970
- Nei sistemi complessi è difficile descrivere il problema come un'unica entità: nella programmazione ad oggetti si decompone il sistema in unità più piccole aventi comportamenti e caratteristiche più semplici: un sistema sw viene realizzato mediante un'insieme di **oggetti che cooperano per la soluzione**. Come nel mondo reale non è necessario conoscerne la struttura interna, ma solo le funzionalità e l'interfaccia di comunicazione

5

OOP-Object Oriented Programming
Introduzione

- Nella OOP non si definiscono i singoli oggetti reali, ma si vanno a individuare le caratteristiche generiche che ogni singolo oggetto deve possedere per far parte di una specifica classe di oggetti

ES. la classe *Processore* è un modello astratto di ogni singolo processore reale. Ogni singolo processore reale è un istanza della classe *Processore*

6

OOP-Object Oriented Programming
Introduzione

- La **programmazione orientata agli oggetti** usa un approccio **buttom-up**, si parte dalle definizione delle classi o dall'individuazioni di classi già esistenti, per creare il programma.
- Migliora:
  - lettura e **comprensione** del codice
  - **Manutenzione**
  - **Riusabilità** del codice in altri programmi
  - **Robustezza** in sistemi complessi o con grandi quantità di dati

7

OOP-Object Oriented Programming
Classe

- Una **classe** è un modello astratto generico per una famiglia di oggetti con caratteristiche comuni: è il suo "template". Ne definisce:
  - le caratteristiche (proprietà o attributi)
  - i comportamenti o funzionalità (metodi)
- Le classi sono organizzate in gerarchie e collegate tramite relazioni di varie tipologie
- L'implementazione di una classe in un linguaggio di programmazione conterrà il codice descrittivo degli oggetti della classe

8

OOP-Object Oriented Programming
Oggetto

- Ogni **oggetto** deriva da una classe (ne è l'istanza)
- Gli oggetti sono in grado di memorizzare uno stato e eseguire operazioni o interagire tra loro richiedendo operazioni ad altri oggetti tramite messaggi

- I metodi e le proprietà sono chiamati anche i **membri** dell'oggetto
- **Stato**: insieme dei valori assunti dalle proprietà in un determinato istante

9

Gli oggetti e le classi

- Una **classe** descrive un insieme di oggetti tramite gli attributi e i metodi che li accomunano, ne rappresenta un modello. Non rappresentano un oggetto particolare. Non può esistere un oggetto se non viene creata la classe a cui appartiene. È la dichiarazione del "tipo", non occupa spazio in memoria
- Gli oggetti creati a partire da una classe sono le **istanze della classe**. Prima si dichiara a quale classe appartengono, poi si alloca lo spazio in memoria e quindi si possono modificare gli attributi o usare i metodi.
- Ogni oggetto della stessa classe si differenzia per il suo stato interno, mentre hanno lo stesso comportamento

Gli oggetti e le classi

- Gli oggetti sono scatole nere la cui unica parte visibile dall'esterno è la sua **interfaccia pubblica**
- Sono gli attori le entità del sistema che si vuole realizzare per risolvere un problema
- Le classi sono "fabbriche" di oggetti: se si ha bisogno di un oggetto si usa una classe come "stampo" per avere un nuovo oggetto con le caratteristiche definite dalla classe
- Si può vedere la classe come un tipo e l'oggetto come una variabile di quel tipo

Classi e oggetti

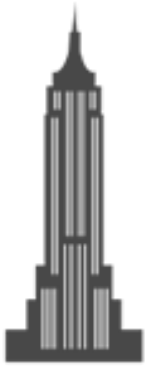
Class
Definition of objects that share structure, properties and behaviours.

Building
class

Dog
class

Computer
class

Instance
Concrete object, created from a certain class.

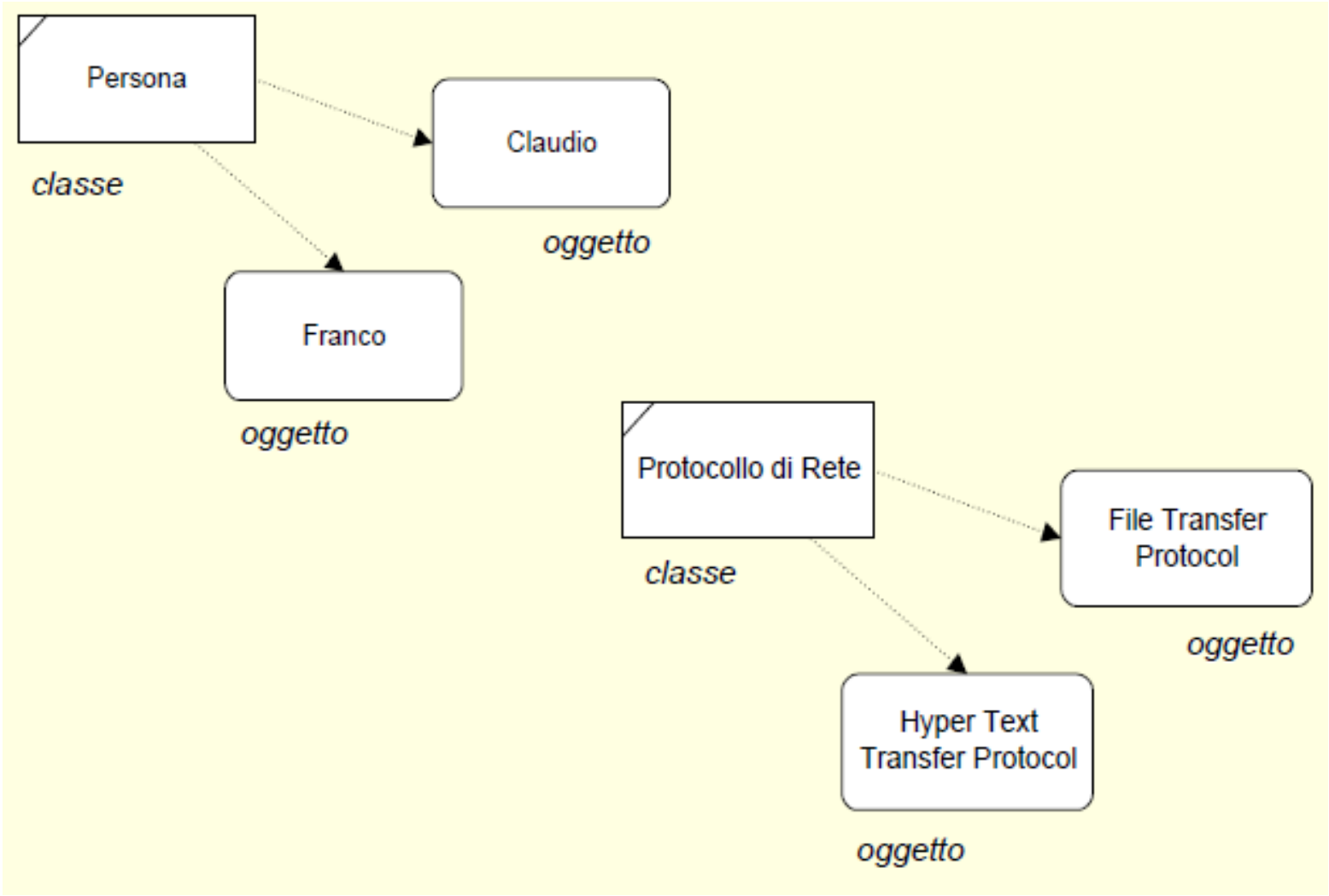


Empire State
instance of Building

Lassie
instance of Dog

Your computer
instance of Computer

Gli oggetti e le classi



Programmare ad oggetti

- Dato il problema si deve:
  - Individuare gli oggetti che realizzano un modello del problema
  - Definire le classi degli oggetti indicando attributi e metodi e relazioni fra classi
  - Stabilire come gli oggetti interagiscono (creando un flusso di messaggi)

14

I messaggi

- Gli oggetti interagiscono scambiandosi **messaggi** (per richiedere dei dati, per attivare un metodo, per cambiare lo stato)
- È costituito da
  - *Destinatario* (nome dell'oggetto)
  - *Selettore* (nome del metodo)
  - *Elenco argomenti* (insieme dei parametri per il metodo)

  Formato messaggio: **oggetto1.metodoX**(a,b,c)
- Quando un oggetto riceve un messaggio verifica se al suo interno è definito il metodo richiesto, se lo è esegue il codice associato

15

OOP-Object Oriented Programming
Linguaggi orientati agli oggetti

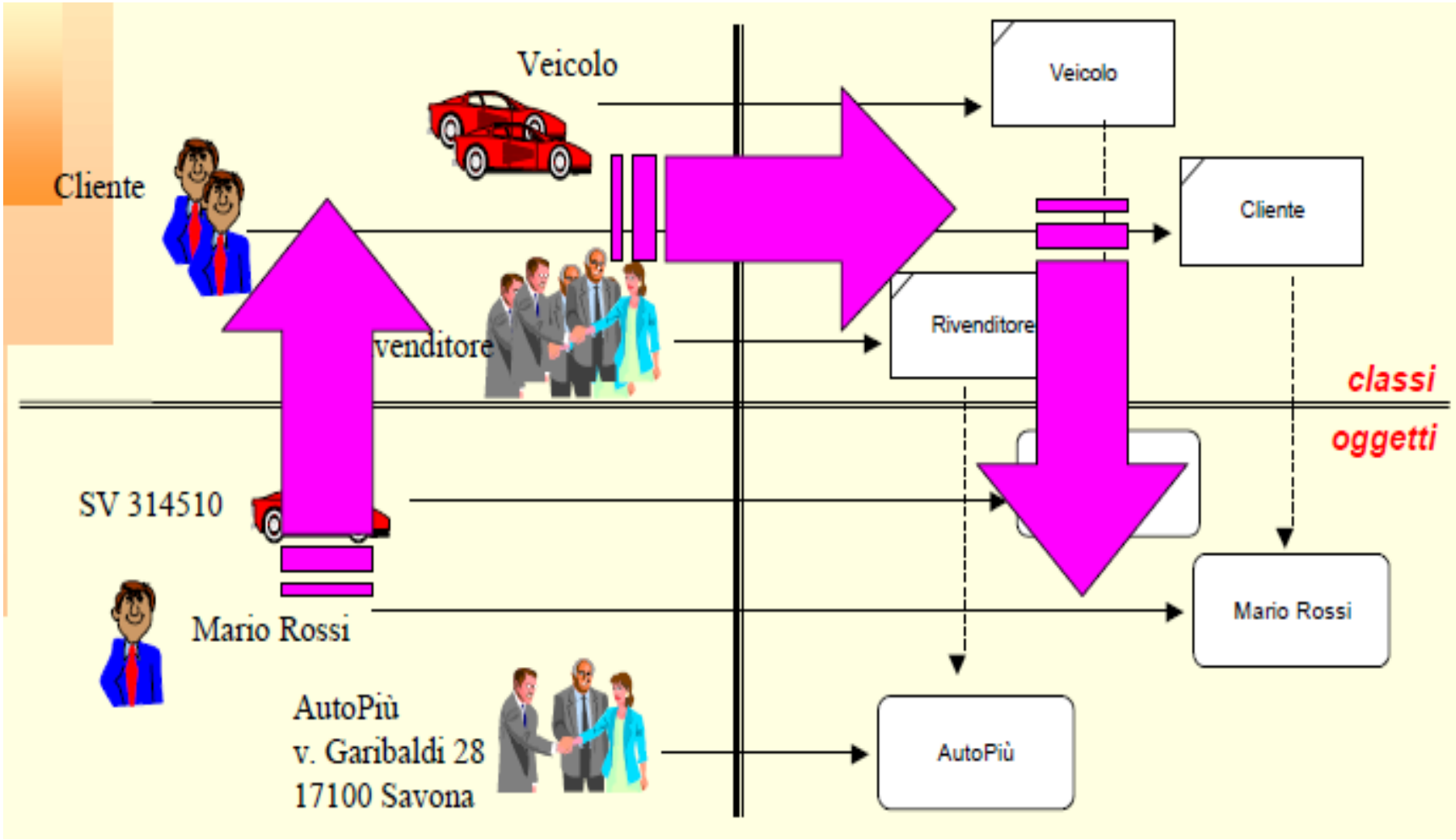
I linguaggi che utilizzano gli oggetti si possono classificare in

Tipo	Permettono di trattare
object-oriented	–classi –oggetti –ereditarietà
class-based	–classi –oggetti
object-based	–oggetti

Quelli **object-oriented** si dicono **puri** se ogni cosa è un oggetto, **ibridi** se è possibile utilizzare tipi di dati "classici

16

Rappresentazione classi e oggetti



17

Tipo di dato astratto

- Un **tipo di dato astratto ATD** è un tipo di dato completamente specificato, ma indipendente dalla sua implementazione. È caratterizzato da:
  - Definizione di un nuovo tipo
  - Definizione delle operazioni possibili sul tipo di dato (**l'interfaccia**)
  - L'unico modo di interagire con il dato è tramite le operazioni previste dall'interfaccia

  Es. tipo frazione algebrica

Tipo di dato astratto e classi

- Le classi realizzano un tipo di dato astratto in quanto definiscono per gli oggetti
  - **Attributi o campi o proprietà**: descrivono l'oggetto tramite delle sue caratteristiche (*caratteristiche statiche*, definiscono lo **stato interno** dell'oggetto). Sono delle variabili
  - **Metodi**: operazioni che sono in grado di svolgere, che utilizzano i suoi attributi (*caratteristiche dinamiche*, definiscono il comportamento dell'oggetto). Sono delle funzioni o procedure accessibili solo attraverso l'oggetto. Per ciascuno deve essere definita **l'*interfaccia o firma*** (nome metodo, numero e tipo/ordine dei parametri) in modo da poter essere richiamato dall'esterno

Visibilità

Package: insieme di librerie di classe correlate, già compilate. Sono organizzati in modo gerarchico, la root è formata dal package *java*. Il package è come una cartella e le classi come i file.

Sono i **namespace** di .net

- La visibilità che una classe ha dei suoi metodi e delle sue proprietà è definita come segue

Visibilità modificatore di accesso	Public	Protected	Package (se non specificato)	Private
Stessa classe	Sì	Sì	Sì	Sì
Classe nello stesso package	Sì	Sì	Sì	No
Sottoclassi stesso package	Sì	Sì	Sì	No
Sottoclasse in package diverso	Sì	Sì *	No	No
Non sottoclasse in package diverso	Sì	No	No	No

* è consentito ad una istanza di una classe figlia far riferimento ai metodi e alle proprietà protected implementate nella classe padre ma non è permesso accedere ad essi attraverso una istanza della classe padre.

20

Sezione pubblica o privata

- I metodi o gli attributi direttamente visibili e utilizzabili da altri oggetti, saranno dichiarati PUBLIC. Normalmente lo sono solo i metodi per il principio del data hiding. La sezione pubblica rappresenta l'interfaccia della classe
- I metodi o gli attributi accessibili solo all'interno della propria classe, in quanto sono utilizzati per implementare il proprio comportamento, sono dichiarati PRIVATE. Normalmente lo sono gli attributi, per questi dovranno essere dichiarati dei metodi pubblici per leggerli **get** o scriverli **set**, in modo che gli oggetti li possano usare.

21

I tipi di metodi

- **Query**: restituiscono lo stato interno
- **Modificatori**: modificano lo stato interno
- In particolare i metodi **accessor set/get** modificano/restituiscono le singole proprietà

  Es. setAltezza() getAltezza()
- **Costruttori**: deve avere lo stesso nome della classe e viene eseguito quando si crea un nuovo oggetto. Inizializzano i vari attributi e compiono tutte le operazioni necessarie alla creazione dell'oggetto. Se non definiti, ne esiste uno definito implicitamente che crea un'istanza della classe che inizializza gli attributi ai valori di default per i tipi corrispondenti senza fare controlli.

Incapsulamento

- Gli oggetti hanno la proprietà di contenere nel loro interno tutto ciò che si riferisce ad esso (caratteristiche e comportamenti), come in una capsula che racchiude tutto quello che li identifica: è l'**incapsulamento.**
- l'incapsulamento è il meccanismo su cui la classe basa la sua esistenza e robustezza. Grazie ad esso ogni classe risulta un'entità ben definita e distinta dal resto del codice: i metodi e i dati al suo interno sono separati da quelli delle altre classi, quindi non possono subire interferenze né possono a loro volta interferire in parti esterne alla classe non di loro competenza. Più semplice controllare gli errori.

23

Information hiding

- Quando tramite un messaggio si richiede un metodo, non si conosce e non interessa come questo sia implementato, che variabili utilizzi, questo perché l'oggetto è come una scatola nera (*blackbox*), che nasconde i dettagli delle sue caratteristiche (si parla anche di **information e data hiding**). Mantenendo l'interfaccia invariata, la manutenzione dei programmi è più efficiente e limitata all'interno del singolo oggetto.

Interfaccia di classe: elenco dei metodi pubblici di una classe, ovvero elenco delle funzioni utilizzabili dalle istanze della classe.

24

Incapsulamento + Information hiding

- Lasciano al programmatore il controllo di accesso ai dati
- Impediscono al programma di finire in uno stato strano o indesiderato

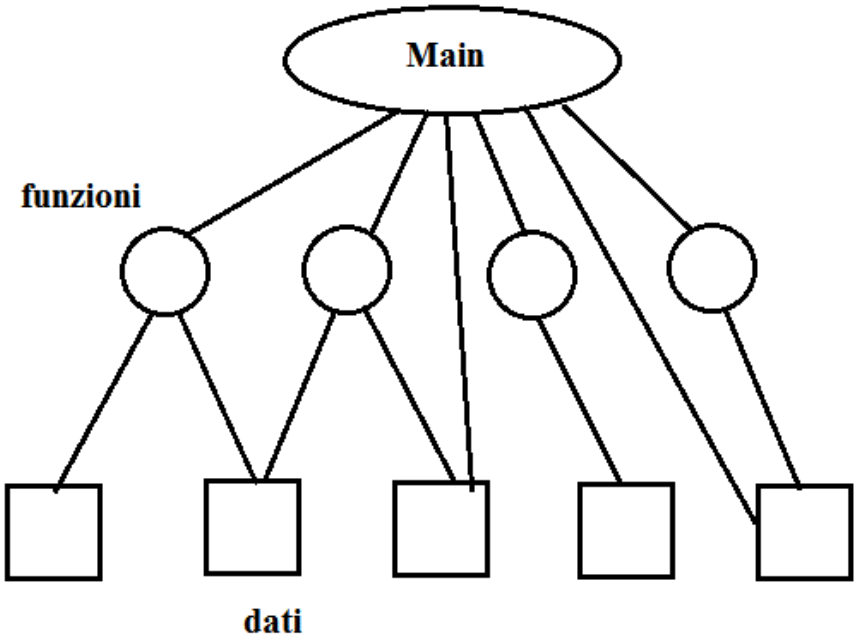
25

Incapsulamento e Information Hiding: regole

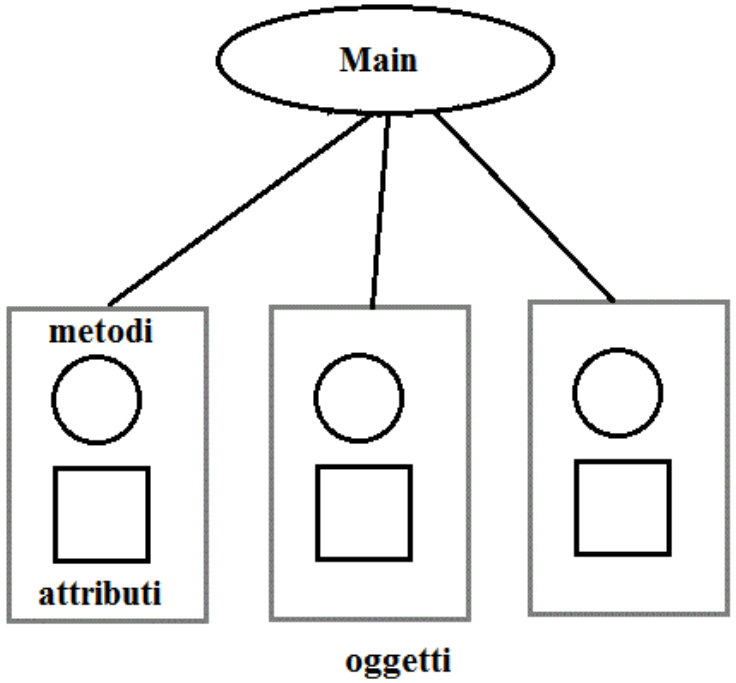
- Incapsulamento
  - Inserire nella stessa classe i dati e le operazioni sui dati
  - Progettare in base alle responsabilità
    - Lasciarsi guidare dalle operazioni: chiedersi quali sono le azioni di cui dovrebbe essere responsabile un oggetto della classe
- Information Hiding
  - Non esporre i dati
    - Dichiarare "privati" tutti gli attributi e usare metodi "get" e "set"
  - Non esporre la differenza tra attributi e dati derivati
    - Se velocità è un dato calcolato da altri attributi chiamare comunque il metodo getVelocità() piuttosto che calcolaVelocità()
- Non esporre la struttura interna della classe
  - Evitare metodi come getDataArray()

26

Programmazione procedurale vs OOP



Programmazione procedurale



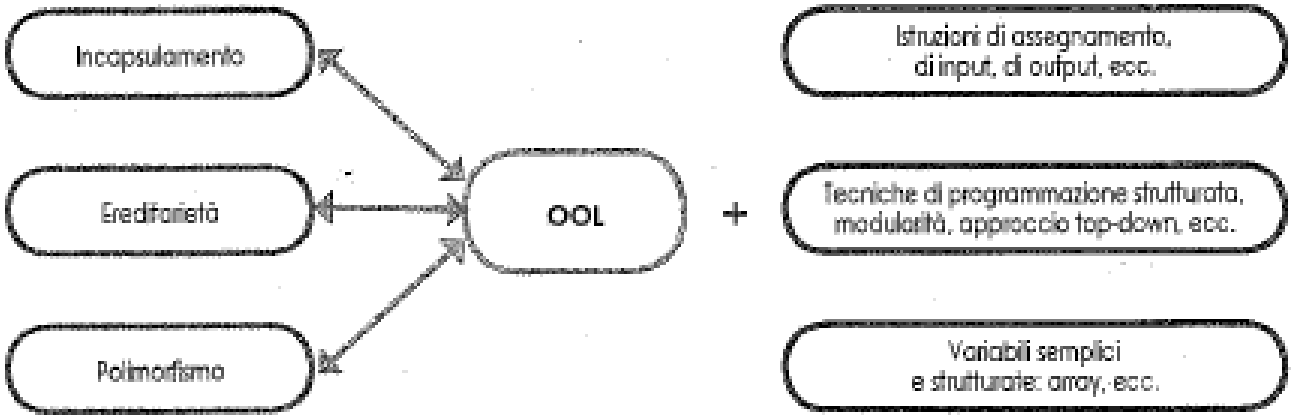
Programmazione ad oggetti

OOP e modularità

Modularità, un concetto da recuperare pienamente

Come abbiamo accennato nella precedente unità, per programmare a oggetti non occorre ricominciare da zero. L'OOP è un'**evoluzione** che spesso integra le buone pratiche di programmazione apprese dal paradigma imperativo con alcuni concetti chiave come incapsulamento, polimorfismo, ereditarietà. Vanno mantenuti, quindi, i concetti di:

- variabili, costanti, tipi;
- istruzioni di assegnamento;
- istruzioni di input, di output,
- tecniche di programmazione strutturata: costrutto di selezione, iterazione;
- variabili strutturate: array,
- concetti chiave, come quello di **modularità**.



Un buon programmatore deve fare attenzione alla complessità di un sistema software. Per risolvere un problema complesso è bene suddividerlo in problemi più semplici e risolvere i singoli problemi. Questa strategia è alla base della modularità.

→ Il concetto di **modularità** consiste nello strutturare l'applicazione software in componenti (o **moduli**) il più possibile indipendenti ma cooperanti tra di loro. In questo modo tali componenti possono essere **riutilizzati** in altre applicazioni.

Il concetto di modularità è alla base della programmazione a oggetti.

Rappresentazione classi e oggetti

- **UML** (Unified Modeling Language) è un linguaggio che permette, tramite l'utilizzo di modelli visuali, di analizzare, descrivere, specificare e documentare un sistema software anche complesso
- **Diagramma delle classi**
  - Struttura logica del sistema
  - Componenti e relazioni tra esse
  - Descrizione generiche di sistema (casi generali)
- **Diagramma degli oggetti**
  - Particolari istanze di sistemi (casi particolari)

29

Formalismo UML

NomeClasse
+attributoPubblico -attributoPrivato #attributoProtetto attributoPackage ±attributoStatico
+metodoPubblico(par1:tipo;par2:tipo2) -metodoPrivato(par1:tipo;par2:tipo2) #metodoProtetto(par1:tipo;par2:tipo2) metodoPackage(par1:tipo;par2:tipo2) *+metodoVirtuale*(par1:tipo;par2:tipo2) ±metodoStatico (par1:tipo;par2:tipo2)

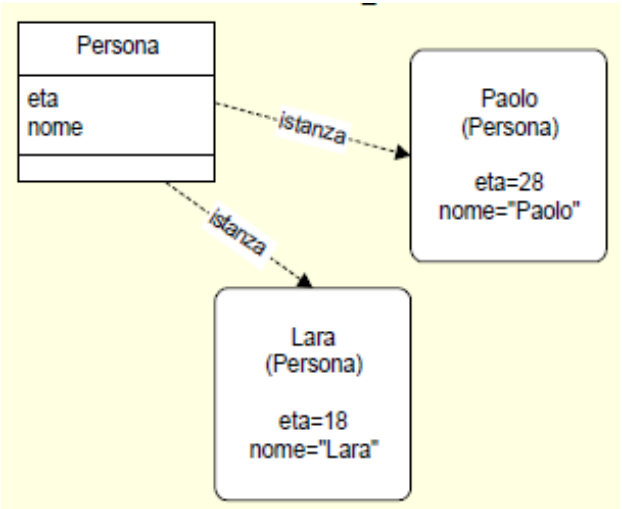
ClassName
Attributo1 : tipo1 Attributo2 : tipo2 = "Valore di Default" ....
operazione1() operazione2(Lista di parametri) operazione3() : Tipo restituito

statico o **di classe**, esiste ed è visibile e utilizzabile anche se non ci sono istanze, è condiviso da tutte le eventuali istanze

30

Rappresentazione classi e oggetti

- Ad ogni **attributo** in una classe corrisponde un valore in un oggetto



- I **metodi** sono uguali per tutti gli oggetti della stessa classe: il comportamento di un oggetto dipende dalla sua classe

31

Occupazione di memoria

In teoria, la generazione di un oggetto per istanzazione di una classe comporta:

- la creazione di una struttura dati analoga a quella definita dalla classe
- la duplicazione nell'oggetto dei metodi specificati nella classe

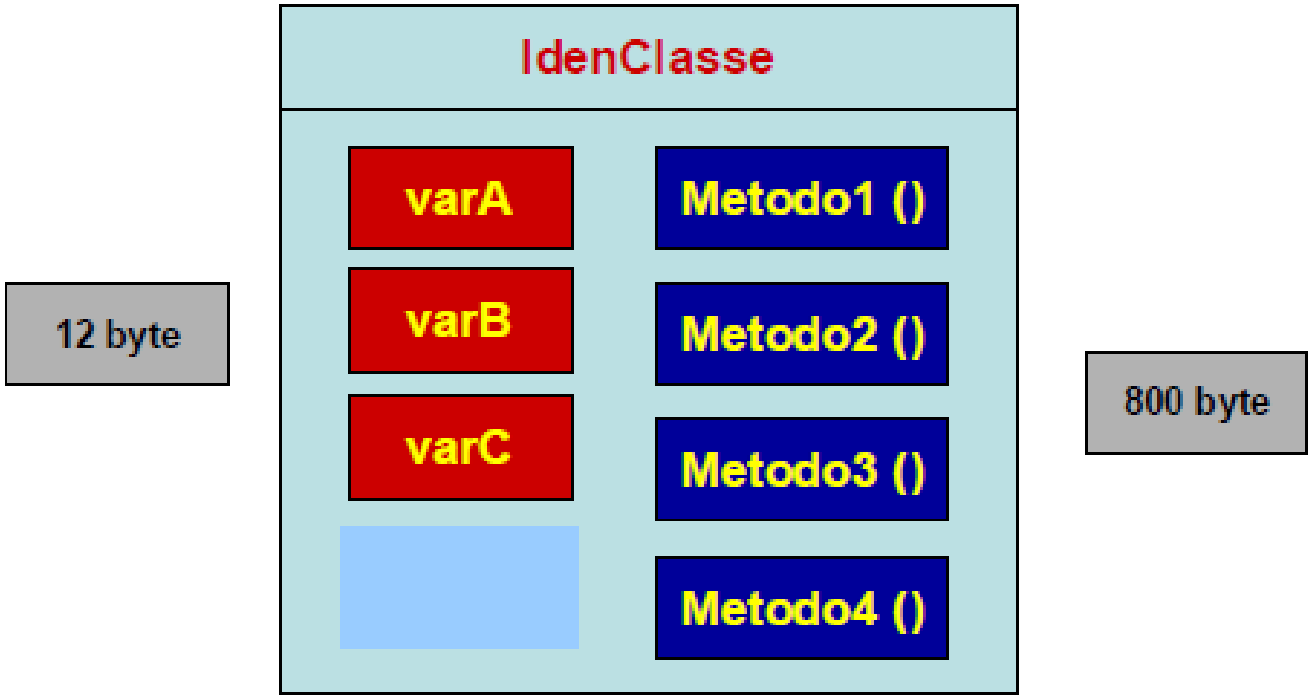
In realtà, è necessario distinguere tra:

- attributi e metodi di istanza (propri dell'oggetto) e di classe (comuni a tutti gli oggetti della classe)
- modello teorico e realizzazione effettiva di classi e oggetti

32

Occupazione di memoria

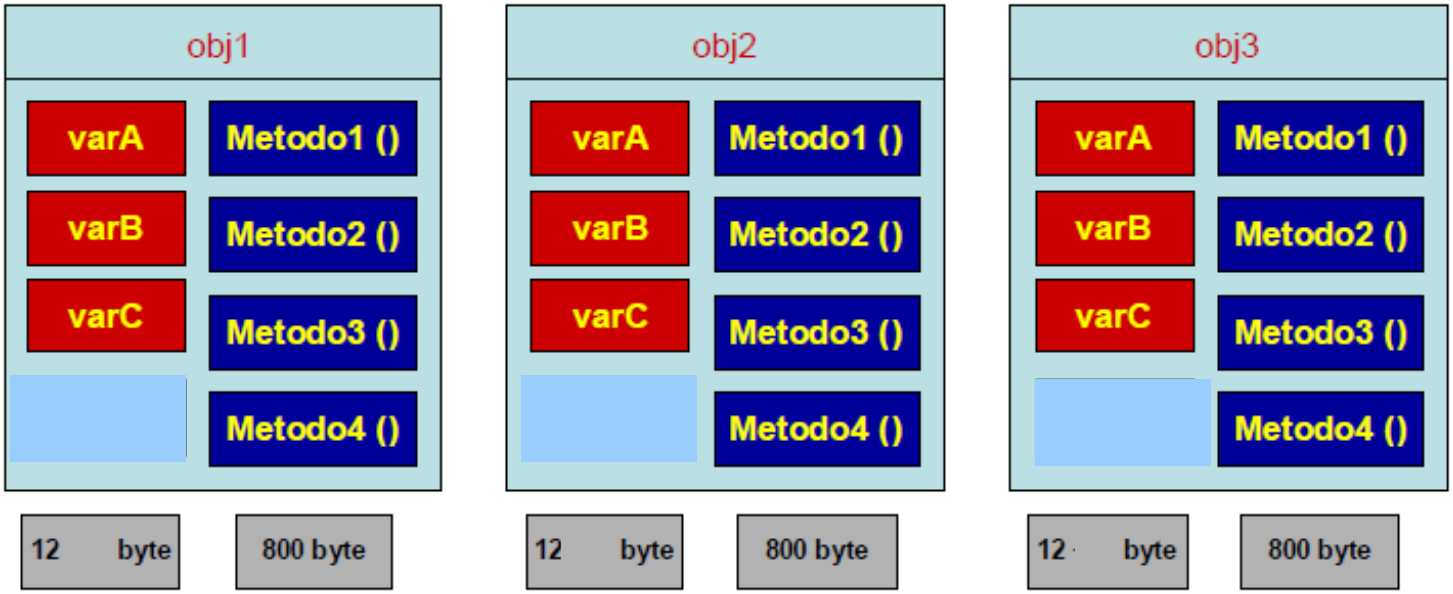
Supponiamo che questa sia la dichiarazione di una classe **IdenClasse**



33

Occupazione di memoria Teoricamente

Da IdenClasse sono istanziati gli oggetti **obj1, obj2, obj3**

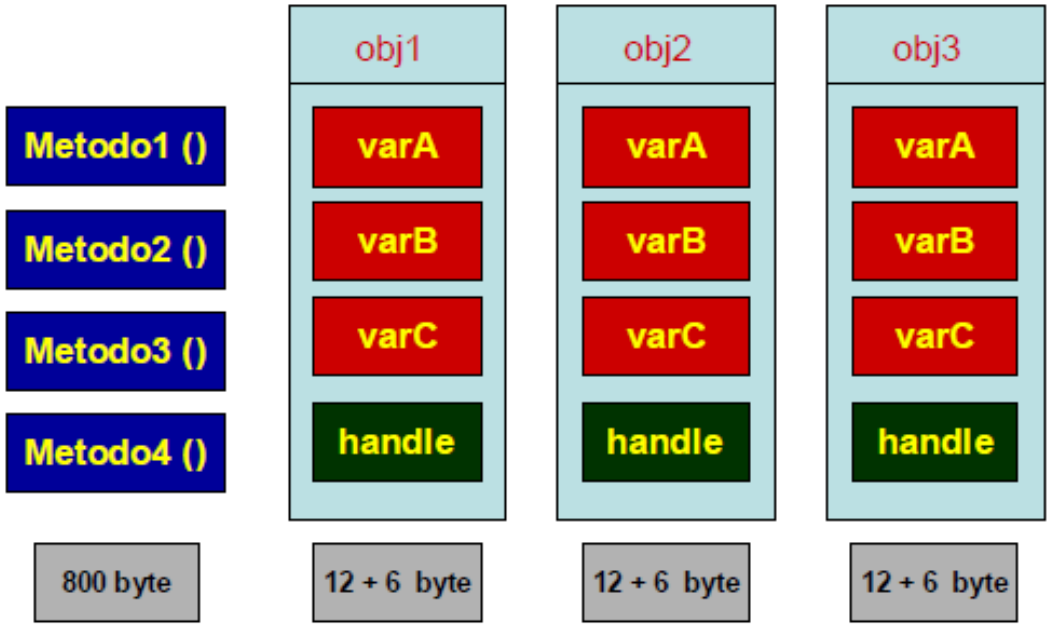


Memoria teorica richiesta: (12+ 800) * 3 = 2436 byte

34

Occupazione di memoria Realizzazione

Da IdenClasse sono istanziati gli oggetti **obj1, obj2, obj3:** duplicando gli attributi e condividendo i metodi (logicamente distinti)

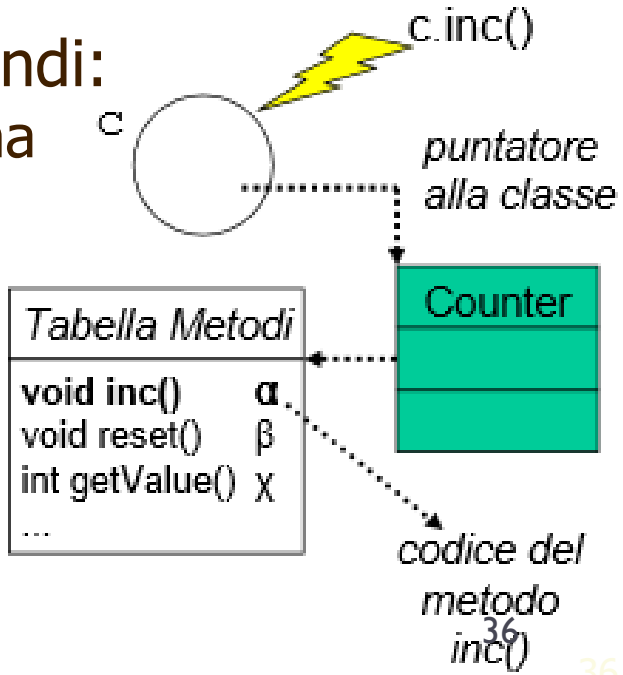


Memoria effettiva richiesta: 800 + 18 * 3 = 854 byte

35

Occupazione di memoria Realizzazione

- Ogni istanza contiene un riferimento alla propria classe che include una tabella VMT (Virtual Method Table) che mette in corrispondenza i nomi dei metodi da essa definiti con il codice compilato relativo a ogni metodo
- Chiamare un metodo comporta quindi:
  - accedere alla tabella VMT opportuna in base alla classe dell'istanza
  - in base alla signature del metodo invocato, accedere alla entry della tabella corrispondente e ricavare il riferimento al codice del metodo
  - invocare il corpo del metodo così identificato



36

Attributi e metodi di classe

- Gli **attributi statici** o **di classe** permettono di definire attributi comuni a tutte le istanze di una classe a cui ci si può riferire anche senza aver istanziato nessun oggetto (viene allocato al primo utilizzo)
- I **metodi statici** o **di classe:** si possono richiamare indipendentemente dall'esistenza di una istanza della relativa classe (sono metodi di utilità). Lavorano solo su attributi statici
- Per riferirsi all'altro tipo di metodi/attributi si parla di **metodi /attributi di istanza**

Ci si può riferire ad essi sia tramite il nome della classe che tramite l'eventuale istanza di essa

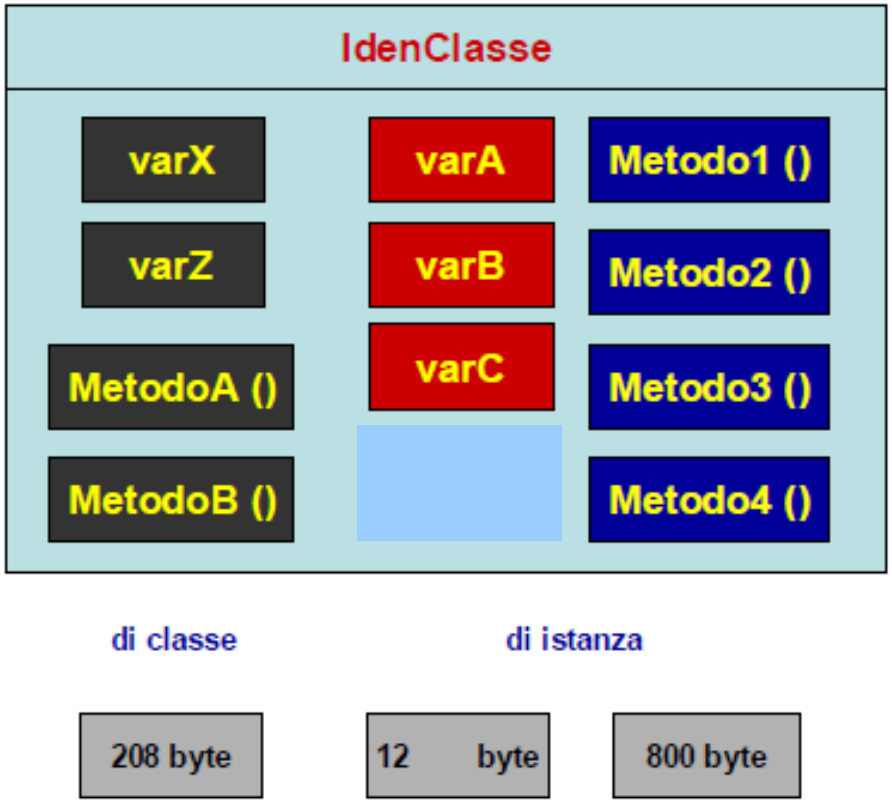
`NomeClasse.attr` oppure `objClasse.attr`

`NomeClasse.met()` oppure `objClasse.met()`

37

Occupazione di memoria Attributi e metodi di classe

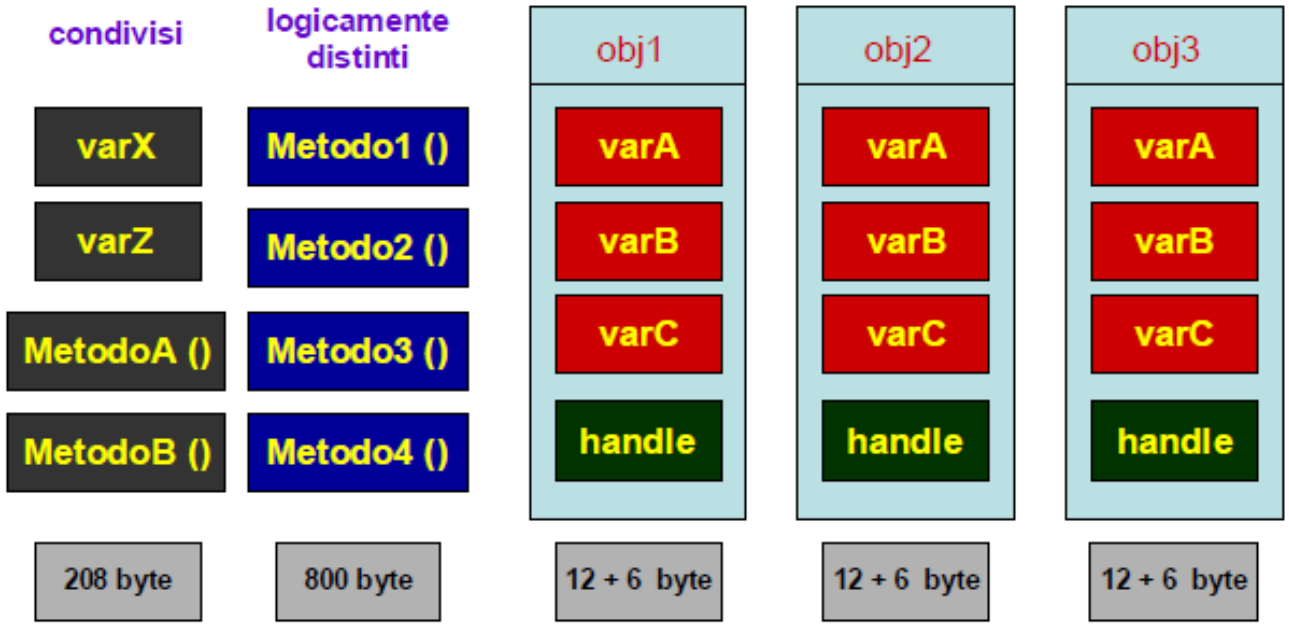
Attributi e metodi (di classe): esiste una copia comune per tutti gli oggetti creati



38

Occupazione di memoria Attributi e metodi di classe

Da IdenClasse sono istanziati gli oggetti **obj1, obj2, obj3** duplicando gli attributi di istanza, condividendo i metodi (logicamente distinti) di istanza e condividendo gli attributi e i metodi di classe



Memoria richiesta: 208 + 800 + 18 * 3 = 1062 byte

39