Corso di Informatica

Modulo T1

B2–Significato e proprietà della OOP

Prerequisiti

- Concetto ed elementi della comunicazione
- Allocazione e deallocazione della memoria
- Compilazione di un programma
- Spazio logico e spazio fisico

Introduzione

La OOP è caratterizzata dal fatto che un programma si compone di oggetti, ciascuno con i propri attributi e metodi.

Un programma produce un'interazione tra gli oggetti, attraverso la chiamata di metodi, e quindi si presenta come una comunicazione tra oggetti.

Come possono gli oggetti di un programma scambiarsi informazioni?

Rispondiamo alla domanda e introduciamo anche le proprietà fondamentali della OOP.

Scambio di messaggi

Un'applicazione **OOP** consiste in uno scambio di messaggi tra oggetti. Ad esempio, per far visualizzare allo **Scrittore** la frase *Ciao a tutti*, è necessario inviargli il seguente messaggio

Scrittore.Stampa ("Ciao a tutti")

- In Java il seguente messaggio produce la stampa della frase

System.out.println ("Ciao a tutti")

Nome dell'oggetto

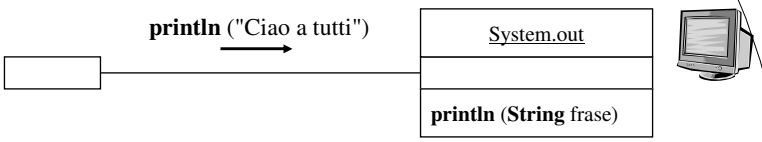
Nome dell'operazione

Parametro

Scambio di messaggi

Un messaggio inviato a un oggetto software specifica una operazione che l'oggetto software deve eseguire, completa di tutti i parametri necessari per l'esecuzione dell'operazione.

Quando un oggetto software riceve un messaggio, allora esegue l'operazione che gli viene richiesta, in modo parametrico rispetto ai dati specificati. Il processo si rappresenta con lo schema grafico seguente **(diagramma di sequenza)**



Comunicazione tra oggetti

Nella programmazione ad oggetti, lo scambio di messaggi che avviene durante l'esecuzione di un programma, stabilisce un'interazione tra oggetti, caratterizzata dai seguenti **elementi della comunicazione**:

- **oggetto mittente** del messaggio (l'oggetto software che invia il messaggio e che potrebbe essere un programma)
- **oggetto destinatario** del messaggio (l'oggetto software a cui viene inviato il messaggio)
- **messaggio** (operazione richiesta al destinatario e scritta mediante una freccia orientata dal mittente al destinatario) con eventuali parametri
- eventuale **oggetto da ottenere in risposta** al messaggio

Comunicazione tra oggetti

Nell'esecuzione di un programma, le interazioni tra oggetti software sono generalmente composte da molti messaggi

Problema: calcolare la radice quadrata del numero 196 e stampare il valore ottenuto.

Per risolvere il problema abbiamo bisogno dell'oggetto **System.out** e dell'oggetto software **Math**:

L'intero processo consiste in uno scambio di due messaggi:

- un oggetto software (il programma) invia a **Math** il messaggio **sqrt**(196). Questo risponde ponendo il valore 14 nella variabile *rad*.
- il programma stesso, successivamente, invia a **System.out** il messaggio **println** *(rad)* per visualizzare il valore calcolato

L'oggetto software Math

L'oggetto software **Math** rappresenta un oggetto che ha capacità di calcolo matematico

Tra le proprietà di **Math** ci sono alcune costanti notevoli (attributi).

oggetto classe <u>**Math**</u>

E = 2.71... // base dei logaritmi naturali
PI = 3.14... // pi greco

double sqrt (**double** n) **double pow** (**double** b, **double** e) **double log** (**double** n) **double random**() ...

Queste sono alcune delle operazioni di **Math** (metodi)

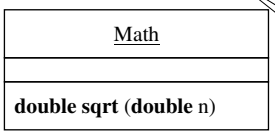
L'oggetto software Math

L'oggetto software **Math** sa eseguire una operazione che calcola la radice quadrata di un numero:

Come sappiamo:

- l'operazione di **Math** per calcolare la radice quadrata di un numero è **sqrt**();
- l'operazione **sqrt**() ha, come parametro, il numero di cui si vuole calcolare la radice quadrata;
- ad esempio, per calcolare la radice quadrata di 144 bisogna scrivere *rad* = **Math.sqrt** (144).

L'oggetto software Math



Quando sull'oggetto **Math** si attiva il metodo **sqrt** (196):

- viene calcolata la radice quadrata di 196 (ossia 14)
- viene posto il valore calcolato nella variabile *rad*

Uso di Math e System.out

A questo punto possiamo rappresentare un processo formato dalle seguenti istruzioni:

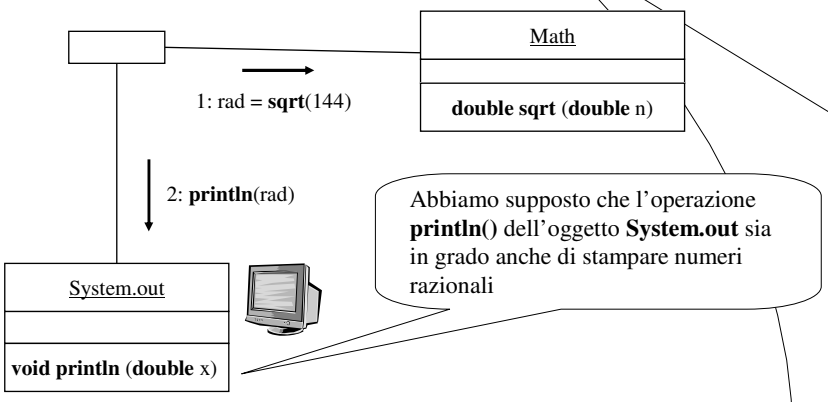
rad = **Math.sqrt** (196);

System.out.println (rad);

- un oggetto software (il programma) chiede a **Math** di attivare il metodo **sqrt**(196). Questo risponde ponendo il valore 14 nella variabile *rad*.
- il programma stesso, successivamente, chiede a **System.out** di eseguire il metodo **println** *(rad)* per visualizzare il valore calcolato.

Uso di Math e System.out

Il processo si rappresenta con diagramma di sequenza seguente:



Esempi diagrammi di sequenza

ATTIVITA': disegnare i diagrammi di sequenza delle seguenti applicazioni:

- Calcolo perimetro di un quadrato nota l'area A
- Portare la velocità dell'automobile da 60 a 30 Km/h e visualizzarla
- Calcolare il punteggio del lancio di 2 dadi e visualizzarlo

Ciclo di vita di un oggetto

Nei linguaggi a oggetti tradizionali (ad es. il C++), il progettista deve saper gestire gli oggetti del programma.

Quando un programma crea un oggetto, questo viene memorizzato in una porzione della RAM detta memoria dinamica, diversa dall'area dati e dall'area programma normalmente usate.

Sulla memoria dinamica, il programmatore agisce:

- allocando spazio per i nuovi oggetti;
- deallocando lo spazio occupato da oggetti che il programma non usa più.

Questa attività fa sì che si possa parlare di **ciclo di vita** di un oggetto, inteso come periodo di permanenza in memoria.

Il *garbage collector*

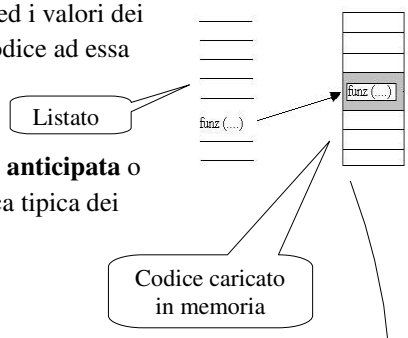
In Java, il problema della gestione della memoria è stato affidato ad un componente della **JVM** detto **garbage collector** che rende l'allocazione della memoria assolutamente trasparente al programmatore.

La **JVM**:

- **alloca la memoria** quando un programma chiede la creazione di un oggetto (operatore **new**);
- **tiene traccia di tutti gli oggetti** e **gestisce gli accessi** che il programma chiede sull'oggetto (accesso agli attributi, esecuzione dei metodi)
- **individua ed elimina dalla memoria**, tutti gli oggetti che non vengono più utilizzati. Di solito, l'individuazione di questi oggetti viene fatta contando il numero di volte che l'oggetto viene referenziato.

Binding statico

Nella programmazione imperativa, il legame tra una funzione ed il suo codice è prestabilito al momento della compilazione. Pertanto, in esecuzione, l'istanza di una funzione, noto il nome della funzione ed i valori dei parametri, produce l'esecuzione del codice ad essa univocamente associato

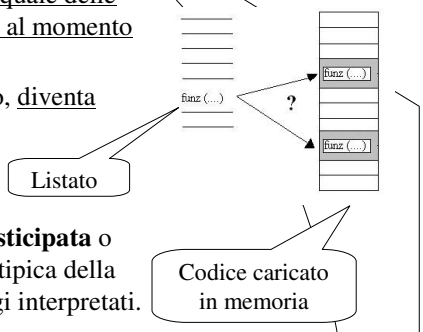


In questo caso si parla di **associazione anticipata** o **binding statico** (*early binding*), tecnica tipica dei linguaggi compilati.

Binding dinamico

In OOP il **polimorfismo** (una proprietà che consente la presenza nella stessa classe di metodi con stesso nome, ma firma diversa) può non rendere chiaro quale delle forme dello stesso metodo verrà istanziata al momento dell'esecuzione.

Il codice che verrà effettivamente eseguito, diventa noto soltanto al momento dell'esecuzione dell'applicazione.



In questo caso si parla di **associazione posticipata** o **binding dinamico** (*late binding*), tecnica tipica della programmazione **OOP** e di molti linguaggi interpretati.

Incapsulamento

Il fatto che attributi e metodi siano incorporati all'interno della classe fa sì che essa si comporti come una **ADT** (*Abstract Data Type*): proprietà e comportamento di un oggetto sono contenuti nell'oggetto stesso e sono visibili soltanto al progettista.

L'utilizzatore ha soltanto accesso alla parte dinamica (metodi) dell'oggetto.

Il progettista associando nella stessa classe attributi e metodi, fa in modo che l'utilizzatore possa agire su essa solo tramite la sua interfaccia.

Questa proprietà si chiama incapsulamento

Incapsulamento

In una classe si possono indicare gli **specificatori di accesso**:

- **public:** contiene i membri accessibili anche all'esterno;
- **private:** non consente l'accesso dall'esterno della classe. La sezione **private** consente l'**occultamento delle informazioni** (*information hiding*) in quanto l'accesso a membri privati può avvenire soltanto mediante metodi implementati allo scopo.

L'information hiding attraverso gli specificatori di accesso, garantisce la protezione dei dati e facilita il riuso del software ed è consentita <u>dall'incapsulamento</u>

Incapsulamento

L'incapsulamento è la prima proprietà fondamentale della **OOP**

INCAPSULAMENTO { **attributi**, **metodi** }

Vantaggi

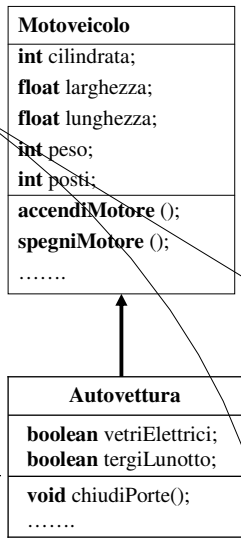
- protezione dei dati
- astrazione sui dati
- astrazione sulle procedure
- mascheramento dell'informazione
- indipendenza degli oggetti
- riutilizzo del software

Ereditarietà

Superclasse

La classe *Motoveicolo*, contiene dei membri che sicuramente faranno parte anche della classe *Autovettura*. Questo vuol dire che potremmo ottenere la classe *Autovettura* per **derivazione** dalla classe *Motoveicolo.* L'ereditarietà è la **seconda proprietà fondamentale della OOP.**

Si possono **aggiungere** membri o **ridefinire** membri: quindi, la <u>sottoclasse diventa una specializzazione della superclasse</u>.



Sottoclasse

Ereditarietà

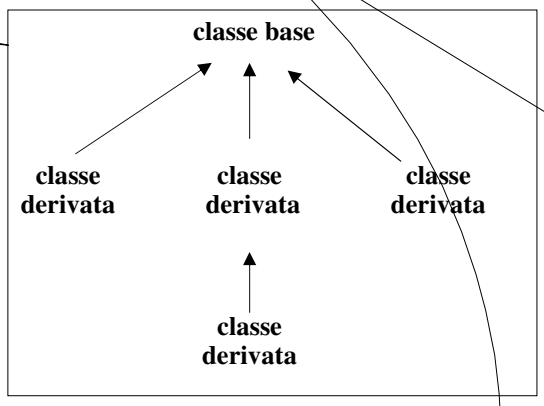
L'ereditarietà è la seconda proprietà fondamentale della **OOP**

Gerarchia di classi

EREDITARIETA'

Vantaggi

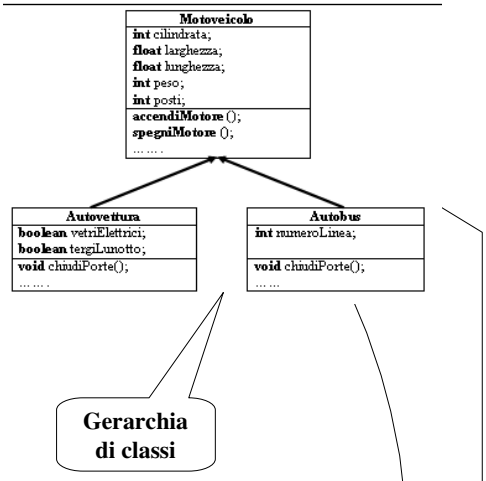
- riutilizzo del software
- espansione del software
- ridefinizione del software



Polimorfismo

Dalla classe *Motoveicolo* deriviamo la classe *Autobus*

Il modo di chiusura delle porte nell'automobile è diverso da quello in un autobus, per cui è necessario fornire nella classe *Autobus* un metodo di chiusura delle porte di nome *chiudiPorte()* che ha lo **stesso nome** di quello della classe *Autovettura.* **Questa proprietà si chiama polimorfismo**.



Polimorfismo

Il polimorfismo è la terza proprietà fondamentale della **OOP**

POLIMORFISMO

- **Metodi omonimi nella stessa classe (overloading)**
- **Metodi omonimi nella stessa gerarchia (overriding)**

Vantaggi

- astrazione procedurale
- riutilizzo del software

Conclusioni

La **OOP** presenta numerosi vantaggi

- **indipendenza tra oggetti** (grazie all'**incapsulamento)** che rende possibile
  - la **riusabilità** del software con un costo minimo di programmazione
  - la **protezione** dei dati (*information hiding)*
- **estensibilità del codice** (proprietà detta **ereditarietà**)
- possibilità che lo stesso codice, in situazioni diverse, produca effetti diversi (proprietà detta **polimorfismo**)

Per brevità d'ora in poi chiameremo semplicemente **oggetto** un qualunque **oggetto software**

Argomenti

- Scambio di messaggi
- Comunicazione tra oggetti
- L'oggetto software **Math**
- Uso di **Math** e **System.out**
- Esempi diagrammi di sequenza
- Ciclo di vita di un oggetto
- Il *garbage collector*
- Binding statico
- Binding dinamico
- Incapsulamento
- Ereditarietà
- Polimorfismo
- Conclusioni

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale 1, ed. Minerva Italica
- M.Romagnoli, P.Ventura – Linguaggio C/C++, ed. Petrini
- A. Garavaglia, F.Petracchi, S.Forte
  Strutture dati e programmazione per oggetti, ed. Masson Scuola