Programmazione I

A.A. 2002-03

Programmazione Orientata agli Oggetti:

Gli oggetti di JAVA (Lezione XXVI)

Prof. Giovanni Gallo
Dr. Gianluca Cincotti
Dipartimento di Matematica e Informatica
Università di Catania
e-mail : {gallo, cincotti}@dmi.unict.it

II

Che si intende per "oggetto"?

Si prenda un oggetto comune quale una CAFFETTIERA. Se vogliamo rappresentarla in un programma dobbiamo descrivere il suo "stato macroscopico" e le operazioni per alterarlo e conoscerlo:

VARIABILI / ATTRIBUTI
per descrivere lo stato della caffettiera

-temperatura nel serbatoio d'acqua;
-quantità d'acqua nel serbatoio;
-quantità di caffè nel filtro;
-quantità di caffè nel bricco;
-temperatura del caffè nel bricco.

METODI
per cambiare lo stato della caffettiera

Aggiungi caffè al filtro;
Aggiungi acqua al serbatoio;
Togli caffè dal bricco;
Aumenta temperatura sul fondo;
Diminuisci l'acqua dal serbatoio e aumenta il caffè nel bricco.

METODI per conoscere lo stato della caffettiera

Fischia?
C'è caffè nel bricco?
Bolle?

Definizione di "oggetto“ in JAVA

Un oggetto è:

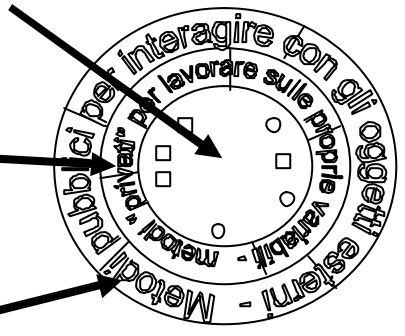
un "contenitore" che tiene assieme:

variabili (o attributi) che descrivono lo stato dell'oggetto o memorizzano dati rilevanti per l'oggetto stesso;

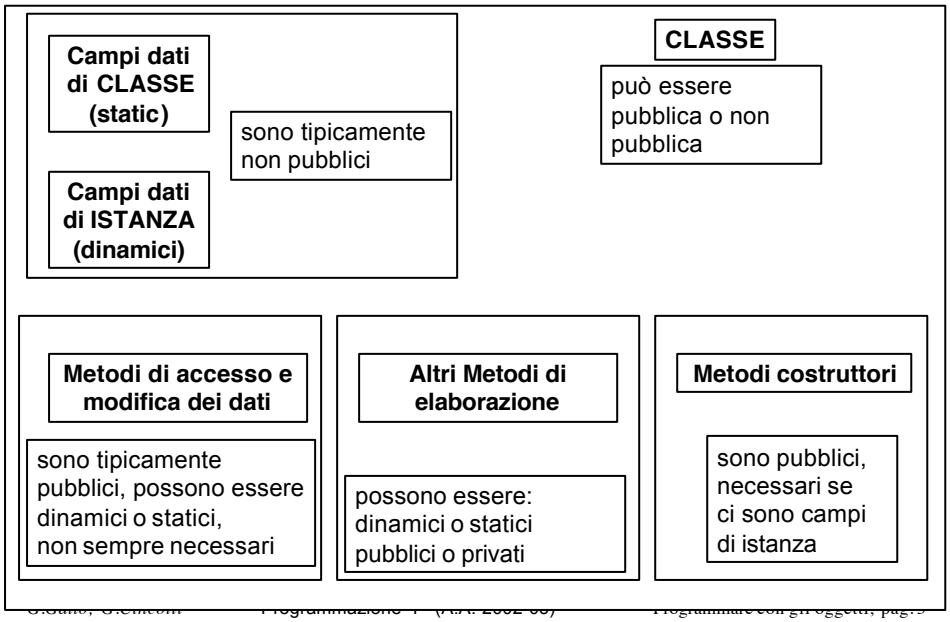
metodi per elaborare le variabili e estrarne, mediante calcoli altre informazioni, oppure per cambiare il valore delle variabili o per "esportare" all'esterno dell'oggetto il loro valore.

La mappa "ideale" di un oggetto

- Nucleo interno "privato" : qui si trovano le variabili che descrivono lo stato dell'oggetto.
- Strato metodi "privati": qui si trovano i metodi privati con cui vengono elaborati i dati privati.
- Strato esterno "pubblico": qui si trovano i metodi o le variabili pubbliche visibili da tutti gli altri oggetti presenti nella JVM.



Struttura di un oggetto



Qualificatori di visibilità dei dati e dei metodi

- public
  vuol dire visibile da tutti gli altri oggetti, indipendentemente dalla classe degli oggetti stessi.
- private
  vuol dire visibile solo da oggetti della medesima classe.
- protected
  Ne parleremo tra un po'.
- non qualificato
  vuol dire visibile solo da oggetti dello stesso "package".

Qualificatori di "modificabilità"

- final

  usato per i campi dati al momento della loro dichiarazione e inizializzazione: indica che il valore del campo non potrà essere alterato in fase di esecuzione.

- final

  usato per le classi, indica che esse non possono essere "estese" (ne parliamo tra un po')

Un oggetto può essere:

? **DICHIARATO, per esempio:**

```
mioTipoOggetto unMioOggetto;
```

? **COSTRUITO, per esempio:**

```
mioTipoOggetto unMioOggetto =
       new mioTipoOggetto(parametri per i campi istanza
                          che si vogliono personalizzare);
```

In entrambi i casi si crea una locazione nella RAM (una variabile) che dovrebbe contenere l' "indirizzo" in RAM dei dati relativi a 'unMioOggetto'.

Nel primo caso però tale indirizzo NON ESISTE, la variabile ha il valore "**null**" (parola riservata di JAVA).

Nel secondo caso la variabile unMioOggetto ha un valore preciso, ma essa è solo l' "indirizzo" dell'oggetto non l'oggetto stesso!

Classi ed Istanze (1)

Una rappresentazione generale è solo "un progetto", una descrizione delle caratteristiche salienti di ogni oggetto di quel tipo.

Un oggetto concreto è una "istanza" particolare che è conforme alla descrizione generica ma è caratterizzata da un preciso valore per le sue variabili, che si riferiscono solo a tale istanza.

ESEMPIO:

La caffettiera: ha un serbatoio per l'acqua,uno per il caffè, un filtro, un coperchio, un manico, valvole di sicurezza eccetera.

La MIA caffettiera: obbedisce a tutte le regole previste dalla descrizione generale della caffettiera.
In più ha: una capacità per i serbatoi (4 tazzine) ed un preciso stato macroscopico in ogni dato istante di tempo della sua esistenza.

Classi ed istanze (2)

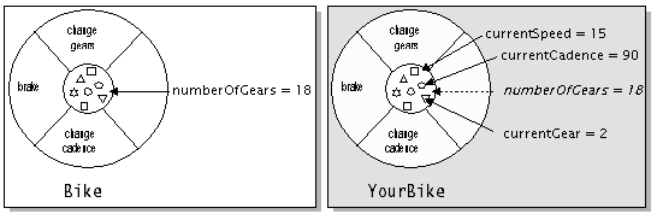
Nel linguaggio di JAVA la descrizione "generale" dell'oggetto si chiama **"class".**

Essa viene creata dal programmatore che "prevede" tutti gli attributi e i metodi che concorrono a formare un oggetto di tale classe.

La realizzazione particolare dell'oggetto descritto dalla classe generale si chiama **"istanza".**

Ogni istanza ha dunque una sua precisa identità e valori propri per ciascuna delle sue variabili in ogni istante della sua esistenza durante la esecuzione del programma da parte della JVM.

Una classe non è un oggetto: esempio



Class **Instance of a Class**

Differenza tra la classe "bike" e la istanza YourBike"

Attributi statici e dinamici

ESEMPIO: Oggetto AUTOMOBILE.

Variabili che ne descrivono lo stato:
numero ruote, numero e funzioni pedali, cilindrata, colore, chilometraggio.

Gli attributi di una classe sono però di due tipi:

Attributi che cambiano valore da istanza ad istanza: si chiamano variabili di istanza o DINAMICHE.

Attributi fissati per tutti gli oggetti della classe: si chiamano variabili di classe o STATICHE.

Ciò è di grande importanza per una implementazione efficiente dentro la RAM!

Variabili e metodi di classe (statiche) e di istanza (dinamiche)

Alcuni attributi (variabili) sono eguali per tutte le realizzazioni pratiche di un oggetto.

Se il valore di una variabile è comune a tutte le istanze della classe dell'oggetto esso può essere definito e fissato una sola volta.
Si tratterà di un attributo di tipo "static".

Non occorre conservare in memoria tante copie dell'attributo quante istanze ci sono di quell'oggetto.

Similmente, alcune funzioni di un oggetto non cambiano da istanza ad istanza: sono le medesime per tutte le istanze. In questo caso i "metodi" che descrivono tali funzioni sono di tipo "static".

Qualificatori di "persistenza"

- Static
  si attribuisce a campi dati o metodi che siano IDENTICI per tutte le istanze di oggetti della classe. Essi non vengono "ricopiati" in RAM ad ogni nuova creazione di oggetti. Di essi si tiene una sola copia!
- Dinamici (non si scrive nulla)
  si attribuisce a campi dati o metodi che possono cambiare da istanza ad istanza di oggetti della medesima classe.

La parola chiave "this"

Questa parola chiave di JAVA viene usata

1. per consentire ad un oggetto di riferirsi ai propri campi e metodi (quando ci potrebbe essere ambiguità o confusione con i campi di un oggetto della stessa classe che è contenuto o usato).

   Esempio 1

2. per riferirsi da dentro un metodo costruttore di un oggetto ad un altro metodo costruttore dell'oggetto stesso.

   Esempio 2

Il codice di "gioco.java": lettura commentata

Rivediamo il codice del "gioco" presentato in precedenza per vedere come le idee sugli oggetti appena visto si traducono nella sintassi di JAVA.

- Regola: in un file si possono definire molte classi, ma una sola è "public" e dà il nome al file.
- In alternativa: ogni classe può essere scritta in un file separato nella medesima directory e compilata separatamente (vedi folder "gioco" dentro il folder "L25").
- L'ordine con cui si definiscono le classi è ininfluente.
- Il main: si tratta di un programma in tutto simile a quelli già visti. La parola chiave "new" serve a costruire una istanza di un oggetto di una classe visibile dalla classe che viene eseguita. Le variabili e i metodi usati dal main debbono essere static! (vedi qualche slide oltre…)

continua ⇒

Il codice di "gioco.java": Le regole di sintassi per gli oggetti

- La class dado: è una classe "dataless" (senza dati). Per tale motivo costruirne una istanza non richiede nessuna operazione e si può fare semplicemente uso del costruttore di default che JAVA ha per tutti gli oggetti (esso carica semplicemente il bytecode nella RAM e conserva riferimento ad esso). Una classe senza dati si può pensare come una "libreria" di metodi che viene utilizzata dalle altre classi.
- Il metodo lancio ( ) della classe dado deve essere "public" per potere essere usato dalle altre class che formano il nostro programma.

continua ⟹

Il codice di "gioco.java": Le regole di sintassi per gli oggetti

- La class giocatore: è la classe più "complessa" (ma sempre molto semplice). Ha un attributo di istanza di tipo int: soldi. Per tale motivo è bene avere un costruttore "ad hoc" che si occupi di tale attributo. Se esso non venisse inizializzato a 10 al momento della costruzione sarebbe inizializzato dal costruttore di default a 0.
- L'attributo di istanza "soldi" è dichiarato "private" per rispettare il principio di encapsulation. Tutti i metodi successivi servono solo a leggerlo e cambiarlo. E' convenzionale chiamare getNomeVariabile il metodo public che "esporta" un attributo e setNomeVariabile il metodo public che ne cambia il valore di NomeVariabile.

continua ⟹

Dentro la JVM (1)

> java gioco

- La JVM viene avviata.
- Si procede a costruire immediatamente l'oggetto "gioco" procedendo in questo ordine:
  - Si riserva un'area di RAM per mantenere la memoria degli attributi di istanza dell'oggetto "gioco" (un giocatore e due dadi).
  - In particolare si riservano come "static" tre locazioni di RAM:
    - una locazione di nome simbolico Pippo conterrà l'indirizzo RAM dei dati relativi ad una istanza della classe "giocatore";
    - due locazioni di nome simbolico dadoBanco, dadoPippo contengono gli indirizzi RAM dei dati relativi a due distinte istanze di oggetti della classe "dado".

L'uso di STATIC vuol dire che tali parole di RAM non saranno cancellate o se non al termine della esecuzione del programma.

Dentro la JVM (2)

- Si costruisce una istanza dell'oggetto giocatore riservando RAM per contenere il valore 10 della variabile di istanza "soldi".
- Vengono inoltre conservati nell'area dell' oggetto Pippo gli indirizzi RAM dove viene ricopiato il bytecode relativo a tutti i metodi della classe giocatore (in memoria si conserva sempre una sola copia di tale bytecode anche se si costruiscono numerose istanze di oggetti della classe).
- L'indirizzo dell'area RAM assegnata all'oggetto Pippo è ricopiato nella corrispondente variabile della classe "gioco".
- Si costruiscono le due istanze dei dadi riservando RAM per ciascuna di esse per contenere gli indirizzi RAM dove viene scritta una unica copia del bytecode relativo all'unico metodo "lancia()" della classe dado.
- L'indirizzo delle aree RAM assegnata all'oggetto dadoBanco e dadoPippo è ricopiato nelle corrispondenti variabili della classe "gioco".

Il bytecode dei metodi di Pippo, di dadoBanco e di dadoPippo non è di tipo "static" e verrà cancellato quando questi oggetti non saranno più usati dal programma.

Dentro la JVM (3)

- Si passa a copiare il bytecode del metodo main di "gioco" in RAM e a mantenerne l'indirizzo. Si richiede che il main sia STATIC cioè esso non potrà essere cancellato dalla RAM fino al completamento del programma stesso.
- Da questo punto in poi tutto è al suo posto: la JVM sa dove trovare i dati e dove il bytecode di tutti i metodi degli oggetti dichiarati. Si passa al ciclo "fetch-decode-execute" del bytecode…

La RAM occupata da ciascun oggetto viene liberata automaticamente dalla JVM quando non ci sono più riferimenti (cioè variabili che si riferiscono) a tali oggetti. Nel nostro semplice caso per tutti i nostri oggetti ciò avviene alla fine della esecuzione del programma.

Classi eseguibili

- Sono eseguibili (cioè JVM le usa come "registi" di una programma ad oggetto) tutte le classi con un metodo statico di nome main.
- Sono eseguibili dal Java Engine dei browser le classi derivate dalla classe "applet" .
- Sono eseguibili da un programma server le classi derivate dalla classe "servlet"

Noi ci occuperemo, in questo corso, solo del primo tipo.

Classi non eseguibili

- Le classi prive del metodo main non sono immediatamente eseguibili. Esse possono soltanto essere "usate" dalle classi con il main.
- E' possibile "aggiungere" un metodo main anche a classi che non lo hanno. Ciò non cambia per nulla le loro funzionalità, essi potranno essere ancora usati da altre classi con main. Il loro main semmai può essere usato per la fase del loro "testing" sconnesse dalle altre classi.

gioco2.java

Fine