Programmazione Orientata agli Oggetti in Linguaggio Java

Classi e Oggetti:
Metafora
Parte b

versione 2.2

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Oggetti
- Costruzione degli Oggetti
  - ⇨Costruttori
  - ⇨Riferimenti
  - ⇨Garbage Collection
- Dati di un Componente

Oggetti

- Riconsideriamo la calcolatrice
  - ⇨una soluzione migliore basata su oggetti
  - ⇨la nuova versione utilizza per le elaborazioni esclusivamente componenti di tipo oggetto
  - ⇨in questa versione il compito delle classi è esclusivamente quello di creare gli oggetti
  - ⇨oltre che di eseguire il metodo main()

>> calcolatrice

Oggetti

- Oggetto
  - ⇨componente creato da un componente classe (istanza della classe)
  - ⇨una classe può creare a richiesta un numero arbitrario di oggetti
  - ⇨tutti gli oggetti creati da una stessa classe hanno caratteristiche simili
  - ⇨sono tutti dello “stesso tipo”

Oggetti

- Metafora
  - ⇨le classi “generano” oggetti (o più correttamente, li “fabbricano”)
  - ⇨un oggetto è il prodotto di una classe
  - ⇨tutti gli oggetti fabbricati da una classe – avendo la stessa origine – hanno caratteristiche simili
  - ⇨ovvero hanno le stesse capacità

Oggetti

- Attenzione
  - ⇨per poter costruire gli oggetti, la classe deve conoscerne la struttura
- Struttura degli oggetti
  - ⇨proprietà e metodi
  - ⇨le proprietà e i metodi degli oggetti vengono descritte nel codice sorgente della classe
  - ⇨proprietà e metodi NON statici

Oggetti

- Metafora
  - ⇨la classe contiene le istruzioni di montaggio per gli omini che fabbricherà
- Ovvero
  - ⇨la classe contiene una descrizione del tipo degli oggetti
  - ⇨la classe definisce il tipo dell'oggetto
  - ⇨gli oggetti sono analoghi a strutture di dati tutte dello stesso tipo

Oggetti

- Come ogni componente, un oggetto ha
  - ⇨un identificatore
  - ⇨un insieme di proprietà (informazioni)
  - ⇨un insieme di metodi (servizi)
- Identificatore di Oggetto (OID)
  - ⇨codice numerico
  - ⇨corrisponde all'indirizzo dell'oggetto nello heap

Oggetti

- Attenzione
  - ⇨differenza importante rispetto alle classi
- Identificatore di un componente classe
  - ⇨nome della classe
  - ⇨ha visibilità globale nell'applicazione
- Identificatore di un componente oggetto
  - ⇨codice numerico
  - ⇨è noto solo a chi ne richiede la creazione

Oggetti

- Creazione di un oggetto
  - ⇨servizio fondamentale offerto dalle classi
  - ⇨equivale ad eseguire un metodo particolare della classe, detto costruttore
- Costruttore
  - ⇨metodo che genera in memoria un oggetto
  - ⇨e ne restituisce l'identificatore (ovvero l'indirizzo di memoria corrispondente)

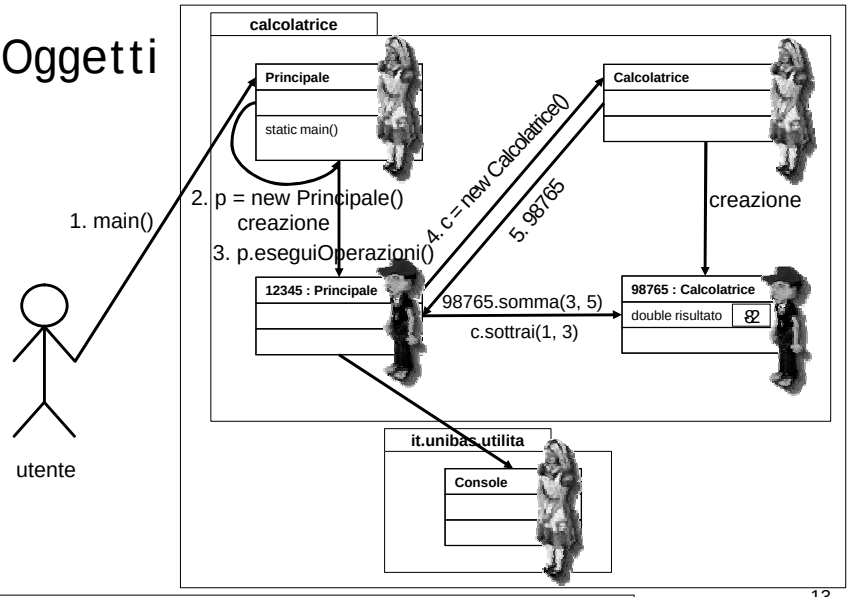
Oggetti

- Strumento per manipolare gli oggetti
  - ⇨variabili di tipo riferimento
- Riferimento
  - ⇨variabile utilizzata per manipolare un oggetto
  - ⇨assume come valore l'identificatore dell'oggetto
  - ⇨funge da "intermediario" verso l'oggetto
  - ⇨in altri termini, consente di inviare messaggi all'oggetto

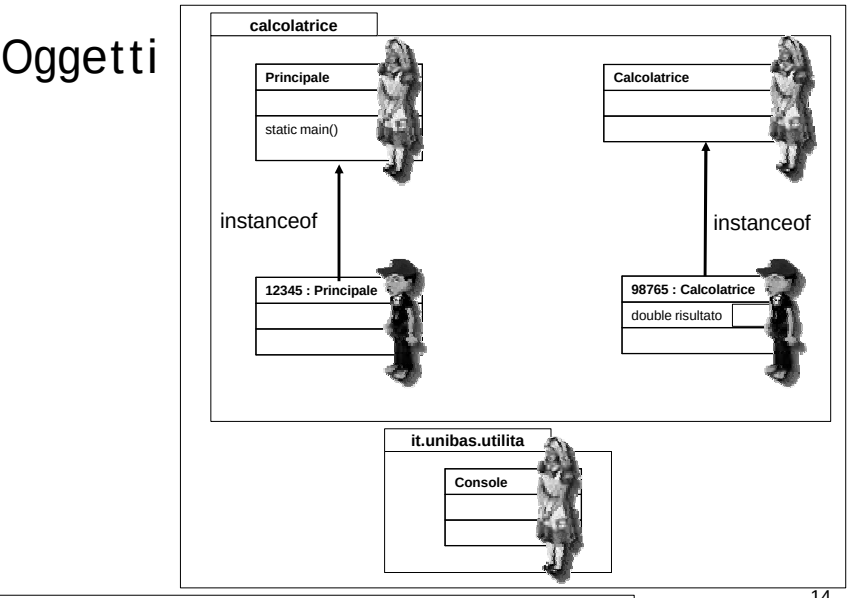
Oggetti

- Operazioni su un riferimento
  - ⇨normalmente, invio di messaggi all'oggetto
- Metafora
  - ⇨il riferimento è il telecomando per pilotare l'oggetto
  - ⇨"premendo i tasti del telecomando", si impartiscono istruzioni al robot

Oggetti



Oggetti



Oggetti

ATTENZIONE
alle differenze
nell'invio di messaggi

- Per inviare un messaggio ad un oggetto
  - ⇨bisogna conoscerne l'id
  - ⇨e cioè possedere un riferimento all'oggetto "telecomando" (visibilità locale)
  - ⇨un riferimento non è qualificato con il nome del package
- Per inviare un messaggio ad una classe
  - ⇨basta usare il nome qualificato (dal package)
  - ⇨il nome qualificato ha visibilità globale

Oggetti

- Metafora
  - ⇨per inviare un messaggio ad una classe basta "urlarne" il nome
  - ⇨tutti i componenti conoscono i nomi di tutte le classi e possono "chiamarle"
  - ⇨non appena una classe si "sente chiamare" risponde al messaggio
  - ⇨per gli oggetti, viceversa, è necessario un "telecomando"

Oggetti

- Rispetto alla soluzione statica
  - ⇨maggior numero di componenti
  - ⇨ma maggiore privatezza nell'elaborazione
  - ⇨l'id dell'oggetto creato dalla classe Calcolatrice è noto esclusivamente all'oggetto della classe Principale
  - ⇨solo quest'ultimo ne può chiamare i metodi
  - ⇨nel caso precedente tutti i componenti potevano cambiare lo stato della calcolatrice

Costruzione degli Oggetti

- Una classe
  - ⇨è un componente (dati e servizi)
  - ⇨ma ha anche compiti speciali
- Compito principale di una classe
  - ⇨creare oggetti
  - ⇨conoscere proprietà statiche
  - ⇨eseguire metodi statici
  - ⇨tra cui l'eventuale metodo statico main

Costruzione degli Oggetti

- E la distruzione degli oggetti ?
  - ⇨dopo averli creati sarebbe opportuno preoccuparsi di “distruggerli” quando non servono più
  - ⇨per evitare di sovraffollare la memoria
- In Java la distruzione è automatica
  - ⇨gli oggetti vengono distrutti automaticamente dal garbage collector della macchina virtuale

Costruzione degli Oggetti

- Nel seguito
  - ⇨rivediamo i tre concetti principali introdotti in questa lezione
  - ⇨costruttori
  - ⇨riferimenti
  - ⇨garbage collector

Costruttori

- Costruttore
  - ⇨metodo con il nome della classe
- Come distinguere un costruttore ?
  - ⇨non viene indicato il tipo di ritorno
  - ⇨non viene indicato static (anche se si può pensare che sia un metodo della classe)
  - ⇨normalmente è public
  - ⇨può o meno avere parametri
  - ⇨es: public Calcolatrice() { ... }

Costruttori

- Costruttore standard
  - ⇨particolare costruttore che (apparentemente) non effettua operazioni
  - ⇨senza parametri (“no-arg”)
  - ⇨corpo vuoto
  - ⇨es: public Calcolatrice() {}
  - ⇨es: public Principale() {}

Costruttori

- Attenzione
  - ⇨tutte le classi devono necessariamente avere almeno un costruttore
  - ⇨può essercene più di uno (fenomeno di sovraccarico)
  - ⇨se il programmatore non ne specifica nessuno, il compilatore aggiunge automaticamente il costruttore standard

Costruttori

- Semantica del costruttore
  - ⇨anche se il costruttore è apparentemente vuoto la macchina virtuale esegue molte operazioni
  - ⇨nella memoria (“heap”) viene allocato lo spazio di memoria per l’oggetto
  - ⇨infine, il costruttore restituisce l’id numerico dell’oggetto, e cioè l’indirizzo del primo registro assegnato all’oggetto

Riferimenti

- Variabile riferimento
  - ⇨spazio nella memoria utilizzato per conservare l'id di un oggetto
- Nota
  - ⇨i riferimenti sono "tipati"
  - ⇨ogni variabile riferimento può contenere solo riferimenti ad oggetti di una classe specifica
- Esempio: Calcolatrice c;
  - ⇨riferimento ad oggetti di tipo Calcolatrice

Riferimenti

ATTENZIONE
alla differenza tra oggetto e riferimento

- Attenzione
  - ⇨un riferimento NON è un oggetto
  - ⇨ma una variabile di un tipo assimilabile ad un tipo primitivo (come int, float, double ecc.)
  - ⇨quindi viene dichiarata in modo ordinario
  - ⇨la memoria relativa viene allocata nello stack
  - ⇨mantiene però il valore di un identificatore di oggetto (un indirizzo dello heap)

Riferimenti

- Utilizzo del costruttore
  - ⇨parola chiave new

```
Calcolatrice c;
c = new Calcolatrice();
```

dichiarazione di una variabile riferimento ad oggetto di tipo Calcolatrice; inizialmente il suo valore è indefinito

chiamata al costruttore della classe Calcolatrice
valore di ritorno: id dell'oggetto
es: 98765
che viene assegnato al riferimento

in alternativa:

```
Calcolatrice c = new Calcolatrice();
```


Riferimenti

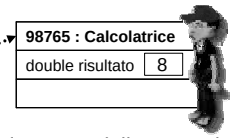
- Semantica

```
public void eseguiOperazioni() {
   double a, b;
   Calcolatrice c = new Calcolatrice();
   ...
   c.somma(a, b);
   ...
}
```

equivale a:
98765.somma(a, b)

#101	a	3
#102	b	5
#103	c	#98765

Pila di attivazione



Altra zona della memoria
("Heap")

Riferimenti

- Valore di una variabile riferimento
  - ⇨tre possibilità
- Valore indefinito
  - ⇨Calcolatrice c;
- Riferimento ad oggetto di tipo opportuno
  - ⇨Calcolatrice c = new Calcolatrice();
- Variabile istanziata al riferimento nullo
  - ⇨Calcolatrice c = null;

Riferimenti

- Operazioni su un riferimento
  - ⇨invio di messaggi ad un oggetto
- Due tipi di messaggi
  - ⇨richiesta di utilizzo di una proprietà
  - ⇨richiesta di esecuzione di un metodo
  - ⇨esattamente lo stesso tipo di messaggi che è possibile inviare ad una classe

Riferimenti

- Utilizzo di una proprietà
  - ⇨*nomeRiferimento.proprietà*
  - ⇨consente di utilizzare la proprietà
  - ⇨es: in Principale

    Calcolatrice c = new Calcolatrice();
    c.risultato = 10;
  - ⇨accettabile solo se le proprietà è pubblica, altrimenti errore a tempo di compilazione
  - ⇨normalmente le proprietà sono private

Riferimenti

- Richiesta di esecuzione di un metodo
  - ⇨*nomeRiferimento.metodo(argomenti)*
  - ⇨apparentemente il programmatore effettua la richiesta direttamente al riferimento
  - ⇨il riferimento si occupa di chiedere all'oggetto l'esecuzione del metodo
- Esempio

  Calcolatrice c = new Calcolatrice();
  c.somma(a, b);

Riferimenti

- Riassumendo
  - la creazione di oggetti è la responsabilità principale delle classi
  - la creazione viene effettuata eseguendo i metodi costruttori
  - la loro funzione è riservare la memoria per un oggetto nello heap e restituirne l'id (indirizzo di memoria)

Riferimenti

- Riassumendo (continua)
  - l'id viene tipicamente assegnato ad una variabile riferimento del tipo opportuno
  - il riferimento viene utilizzato come intermediario per eseguire metodi dell'oggetto
  - i messaggi inviati attraverso il riferimento vengono "inoltrati" ed eseguiti dall'oggetto

Riferimenti

- this: una variabile riferimento particolare
  - ⇨contiene in ogni oggetto il valore del proprio identificatore (non è necessario dichiararla)
  - ⇨può essere utilizzata per richiamare i propri metodi e le proprie proprietà
- Esempio: in Calcolatrice

```
public void moltiplica(double a, double b) {
        this.risultato = a * b;
}
```


Garbage Collection

- Quanto vivono gli oggetti ?
  - ⇨dopo la loro creazione gli oggetti occupano memoria
  - ⇨per recuperare memoria, sarebbe bene liberarsi degli oggetti diventati inutili
- Quando un oggetto è diventato inutile ?
  - ⇨quando non ci sono più in memoria variabili che contengono un riferimento all'oggetto
  - ⇨es: è stato creato in un metodo terminato

Garbage Collection

- In alcuni linguaggi di programmazione
  - ⇨il programmatore deve distruggere esplicitamente gli oggetti quando non servono più
- In Java
  - ⇨la distruzione viene fatta automaticamente dalla macchina virtuale
  - ⇨operazione di “garbage collection”

Garbage Collection

- Garbage collection
  - ⇨periodicamente la macchina virtuale analizza automaticamente lo heap
  - ⇨di ciascun oggetto, tiene traccia di tutte le variabili riferimento
  - ⇨se trova oggetti per i quali non ci sono più riferimenti disponibili, li distrugge e libera la memoria corrispondente

Garbage Collection

- Metafora
  - ⇨come i televisori non possono essere usati senza telecomando, così gli oggetti non possono essere usati senza riferimenti
  - ⇨nella città degli omini c'è un netturbino
  - ⇨che raccoglie e butta via tutti i robot inutilizzabili, ovvero quelli per cui non ci sono più in giro telecomandi disponibili

Dati di un Componente

- Proprietà di un componente
  - ⇨si tratta essenzialmente di variabili
  - ⇨sono le informazioni che il componente deve mantenere per tutto il suo ciclo di vita
  - ⇨ne descrivono lo "stato"
  - ⇨sono visibili in tutti i metodi del componente
- Inoltre
  - ⇨un componente manipola anche altri dati

Dati di un Componente

- Dati utilizzati nei metodi
  - ⇨parametri ed eventuali variabili locali
- Questi dati
  - ⇨non rappresentano proprietà dell'oggetto
  - ⇨hanno visibilità locale al metodo
- Una variabile particolare
  - ⇨il riferimento this (riferimento a sè stesso)
  - ⇨consente l'accesso alle proprietà e ai metodi

Esempio

```
public void moltiplica(double a, double b) {
      this.risultato = a * b;
}
```

- Dati del componente "Calcolatrice"
  - ⇨la calcolatrice ha un'unica proprietà risultato
  - ⇨visibile in tutti i metodi di un oggetto della classe
  - ⇨mantiene lo stato della calcolatrice
  - ⇨necessaria per le operazioni con memoria
- Altri dati utilizzati nei metodi
  - ⇨parametri (es: a, b)
  - ⇨visibili solo nel metodo

ATTENZIONE
è necessario
limitare al massimo
le proprietà dei componenti

Esempio

- Come scegliere le proprietà ?
  - ⇨un oggetto dovrebbe avere il minimo numero di proprietà
  - ⇨tutte e sole le informazioni da conservare durante la vita dell'oggetto
  - ⇨tutto il resto: parametri o variabili locali
- Nell'incertezza
  - ⇨meglio una proprietà in meno che una in più

```
package calcolatrice;
public class Calcolatrice {

   public Calcolatrice() {} // opzionale

   private double risultato;

   public double getRisultato() {
      return this.risultato;
   }

   public void somma (double a, double b) {
      this.risultato = a + b;
   }

   public void sottrai (double a, double b) {
      this.risultato = a - b;
   }

   public void moltiplica (double a, double b) {
      this.risultato = a * b;
   }
   ...
```

```
package calcolatrice;
public class Principale {

   public static void main(String args[]) {
      Principale p = new Principale();
      p.eseguiOperazioni();
   }

   public Principale() {} // opzionale

   public void eseguiOperazioni() {
      Calcolatrice calcolatrice = new Calcolatrice();
      boolean continua = true;
      while (continua) {
         double a, b, risultato = 0;
         int scelta = this.schermoMenu();
         if (scelta == 0) {
            continua = false;
         } else {
            System.out.print("   Inserisci gli argometi:   --> ");
            a = it.unibas.utilita.Console.leggiDouble();
            b = it.unibas.utilita.Console.leggiDouble();
            if (scelta == 1) {
               calcolatrice.somma(a, b);
               risultato = calcolatrice.getRisultato();
            }
         ...
```


Riassumendo

- Oggetti
- Costruzione degli Oggetti
  - ⇨Costruttori
  - ⇨Riferimenti
  - ⇨Garbage Collection
- Dati di un Componente

Termini della Licenza