Programmazione Orientata agli Oggetti in Linguaggio Java

Classi e Oggetti:
Metafora
Parte c

versione 2.1

Questo lavoro è concesso in uso secondo i termini di una licenza Creative Commons
(vedi ultima pagina)

SOME RIGHTS RESERVED
creative commons

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

Sommario

- Scegliere il tipo di Componente
- Analisi di Circonferenze
  - ⇨Metodi di Accesso e Modifica
  - ⇨Costruttori con Argomenti
  - ⇨Costanti
  - ⇨Array
- Compiti dei Componenti

Scegliere il Tipo di Componente

- Riassumendo
  - ⇨esistono due tipi di componenti
  - ⇨classi e oggetti
- Come scegliere tra i due tipi ?
- Regola fondamentale
  - ⇨è preferibile programmare con gli oggetti
  - ⇨e limitare l'utilizzo dei metodi statici

Scegliere il Tipo di Componente

- Compiti delle classi
  - ⇨creare oggetti
  - ⇨eseguire metodi statici (e conoscere proprietà statiche)
  - ⇨tra questi il metodo speciale main()
  - ⇨trascuriamo il metodo main() che ha uno scopo speciale e cerchiamo di classificare le classi

Scegliere il Tipo di Componente

- Tre tipologie di classi
  - ⇨classi che non eseguono metodi statici: eseguono esclusivamente i costruttori
  - ⇨classi che non eseguono costruttori: eseguono esclusivamente metodi statici
  - ⇨classi che eseguono entrambi i servizi
- Tipicamente
  - ⇨le classi dovrebbero essere del tipo a. oppure b. – attenzione al tipo c.

Scegliere il Tipo di Componente

- Metafora
  - ⇨fabbricare oggetti “stanca”
  - ⇨di conseguenza esistono “classi fabbricatrici” >> classi specializzate nel fabbricare oggetti
  - ⇨e “classi di servizio” >> classi specializzate nell’eseguire metodi statici
  - ⇨fare tutte e due le cose non è opportuno (troppo stancante)

Scegliere il Tipo di Componente

- Nota
  - ⇨dalla classificazione escludiamo le costanti (statiche) e il metodo main
- Esempi di classi fabbricatrici
  - ⇨Principale e Calcolatrice in calcolatrice
- Esempi di classi di servizio
  - ⇨it.unibas.utilita.Console
  - ⇨java.lang.System (solo metodi statici)

Scegliere il Tipo di Componente

>> it.unibas.utilita.Console

- Attenzione
  - ⇨tutte le classi hanno almeno un costruttore
  - ⇨quindi normalmente è possibile creare oggetti di tutte le classi
  - ⇨come fare per impedirlo (nel caso di componenti che non devono avere istanze ?)
- Costruttore private
  - ⇨in questo modo la costruzione è impedita al di fuori della classe

Scegliere il Tipo di Componente

- Le classi di tipo c.
  - ⇨classi fabbricatrici e di servizio assieme
  - ⇨eseguono costruttori e metodi statici contemporaneamente
  - ⇨vanno utilizzate con parsimonia
- Quando è opportuno utilizzarle ?
  - ⇨lo vediamo con un altro esempio

Analisi di Circonferenze

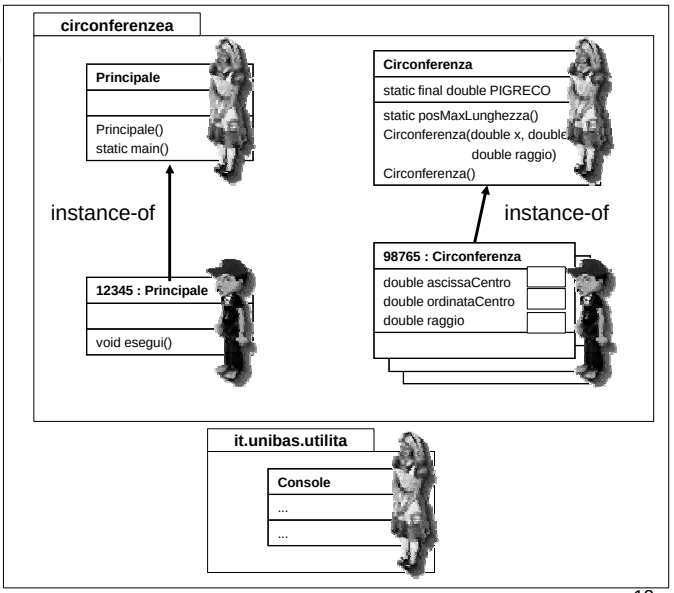
- Un altro esempio: analisi di circonferenze
  - ⇨un programma che acquisisce i dati relativi a varie circonferenze nel piano cartesiano
  - ⇨utilizza un array (>>)
  - ⇨effettua vari calcoli su ciascuna circonferenza e ne stampa i risultati sullo schermo
  - ⇨infine cerca la circonferenza più lunga

>> circonferenzea

Esempio

- Soluzione
  - ⇨un oggetto per ciascuna circonferenza
  - ⇨tutti creati dalla classe Circonferenza
- Ogni circonferenza
  - ⇨conosce le coordinate del suo centro ed il suo raggio
  - ⇨è in grado di calcolare la sua lunghezza e la superficie del suo cerchio

Esempio



Esercizio: disegnare il diagramma che illustra le chiamate dei metodi in questo esempio

Esempio

- Alcune particolarità dell'esempio
  - ⇨metodi di accesso (get) e modifica (set)
  - ⇨costruttore con parametri
  - ⇨la costante PIGRECO
  - ⇨utilizzo degli array
- Li discutiamo nel seguito

Metodi di Accesso e Modifica

- I metodi degli oggetti Circonferenza
  - ⇨la maggior parte comincia con get o con set
- Esempio

```
public double getAscissaCentro () {
   return this.ascissaCentro;
}

public void setAscissaCentro (double ascissaCentro) {
   this.ascissaCentro = ascissaCentro;
}
```

Metodi di Accesso e Modifica

- Idioma: metodi di accesso e modifica
  - ⇨“getters” and “setters”
  - ⇨per ciascuna proprietà privata vengono creati, se necessario, due metodi
  - ⇨un eventuale metodo per rendere visibile all’esterno il valore della proprietà (“getter”)
  - ⇨un eventuale metodo per modificare dall’esterno il valore della proprietà (“setter”)

Metodi di Accesso e Modifica

```
public class Circonferenza {
    private double raggio;

    public double getRaggio() {
        return this.raggio;
    }

    public void setRaggio(double raggio) {
        this.raggio = raggio;
    }

    ...
}
```

proprietà (privata)

metodo “getter” get + nome della proprietà con la maiuscola (notazione cammello) - senza parametri

metodo “setter” set + nome della proprietà con la maiuscola (notazione cammello) - un parametro

Metodi di Accesso e Modifica

- Un'annotazione
  - ⇨non sempre i metodi il cui nome rispecchia quello di un getter o di un setter corrispondono ad una proprietà
- Esempio
  - ⇨public double getLunghezzaCirconferenza() {...}
  - ⇨public double getSuperficieCerchio() {...}

Metodi di Accesso e Modifica

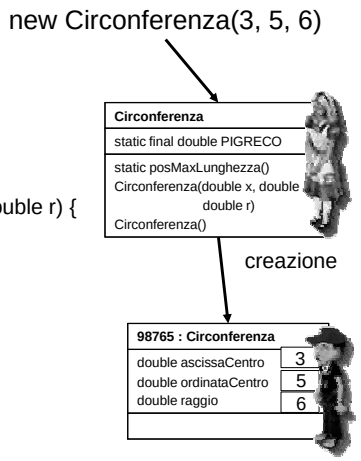
- Infatti
  - ⇨le proprietà sono solo le coordinate del centro ed il raggio (informazioni indispensabili per rappresentare la circonf.)
  - ⇨la lunghezza della circonferenza e la superficie del cerchio NON sono proprietà
  - ⇨perché possono essere facilmente calcolate a partire dalle proprietà

Costruttore con Parametri

- Costruttore con parametri
  - ⇨una classe può avere più di un costruttore
  - ⇨i vari costruttori devono distinguersi sulla base del prototipo
  - ⇨fenomeno di sovraccarico
- Nel caso di Circonferenza
  - ⇨public Circonferenza() {}
  - ⇨public Circonferenza(double x, double y, double r)

Costruttore con Parametri

```
public class Circonferenza {
   ...
   private double ascissaCentro;
   private double ordinataCentro;
   private double raggio;
   public Circonferenza(double x, double y, double r) {
      raggio = r;
      ascissaCentro = x;
      ordinataCentro = y;
   }
   ...
```



Costruttore con Parametri

- Attenzione
  - ⇨i seguenti frammenti sono equivalenti

```
Circonferenza c = new Circonferenza (3, 5, 8);
```

```
Circonferenza c = new Circonferenza ();
c.setAscissaCentro(3);
c.setOrdinataCentro(5);
c.setRaggio(8);
```


Costruttore con Parametri

- Vantaggi della prima forma di costruttore
  - ⇨quando l'oggetto viene costruito è già in uno stato "consistente" (non è necessario dover eseguire i metodi set)
  - ⇨se l'oggetto è "immutabile" è possibile omettere i metodi set e così evitare che lo stato sia cambiato per errore
- Vantaggio della seconda forma
  - ⇨il costruttore può essere omesso

Costruttore con Parametri

- In generale
  - ⇨la prima forma ha dei vantaggi
- Ma, in alcuni contesti
  - ⇨è indispensabile utilizzare la seconda forma
  - ⇨es: utilizzo di JavaBeans in contenitori
  - ⇨o è vantaggioso usare la seconda forma (es: gerarchie complesse di ereditarietà)

Costanti

- Altra particolarità
  - ⇨la costante PIGRECO
  - ⇨public static final double PIGRECO = 3.14;
- Qualificatore final
  - ⇨stabilisce che la risorsa a cui è applicata è immutabile
  - ⇨normalmente le costanti sono globali e pubbliche (public static)

Array

- Array in Java: concetti familiari
  - ⇨un array è una collezione di variabili tutte dello stesso tipo
  - ⇨l'array ha una dimensione fissata e la memoria viene assegnata staticamente
  - ⇨le componenti si accedono utilizzando la sintassi delle parentesi quadre; es: collezione[i]

Array

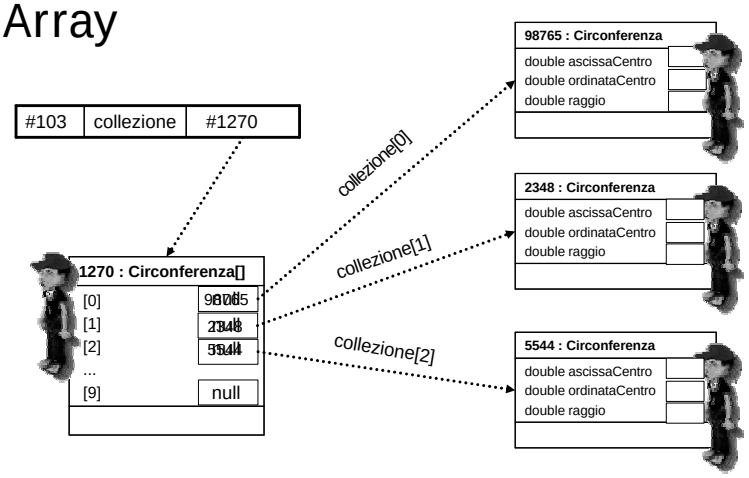
- Ci sono però delle differenze
  - ⇨un array Java è un tipo particolare di oggetto; viene creato usando la parola chiave new
  - ⇨come tutti gli oggetti viene manipolato attraverso un riferimento
  - ⇨la dimensione non deve essere per forza una costante (può essere una variabile)
  - ⇨ogni array ha una proprietà pubblica length che contiene la sua dimensione

Array

- In Java
  - ⇨l'array può contenere variabili di tipi di base
  - ⇨oppure variabili riferimento
- Nel nostro caso
  - ⇨un array di riferimenti a oggetti di tipo Circonferenze

    Circonferenza[] collezione = new Circonferenza [10];
  - ⇨inizialmente tutti i riferimenti sono null

Array



Array

- In altri termini
  - ⇨esistono due tipologie di array completamente diverse l'una dall'altra
  - ⇨gli array di dati primitivi sono componenti assimilabili agli array tradizionali
  - ⇨gli array di riferimenti sono collezioni di riferimenti
- Metafora
  - ⇨una collezione di riferimenti è un supertelecomando
  - ⇨che consente di utilizzare molti componenti diversi

Array

```
public void esegui() {
      int numCirconferenze = schermoLeggiNumCirconferenze();
      if (numCirconferenze > 0) {
         Circonferenza[] collezione = new Circonferenza[numCirconferenze];
         for (int i = 0; i < collezione.length; i++) {
            collezione[i] = schermoLeggiDatiCirconferenza();
         }
         for (int i = 0; i < collezione.length; i++) {
            schermoStampaDatiCirconferenza(collezione[i]);
         }
         System.out.println("\nLa circonferenza di lungh. massima è in pos: " +
                           Circonferenza.posMaxCirconferenza(collezione));
      }
      System.out.println("\nArrivederci.");
   }
```

Compiti dei Componenti

- La classe Principale
  - ⇨è una classe fabbricatrice
  - ⇨ed inoltre esegue il metodo main
- L'oggetto di tipo Principale
  - ⇨esegue gli schermi
  - ⇨gestisce il flusso di controllo dell'applicazione
  - ⇨crea le varie Circonferenze

Compiti dei Componenti

valore di ritorno (riferimento ad una circonferenza)

```
private Circonferenza schermoLeggiDatiCirconferenza() {
    System.out.println("\nImmetti i dati della circonferenza:");
    System.out.print("Ascissa del centro: --> ");
    double ascissaCentro = it.unibas.utilita.Console.leggiDouble();
    System.out.print("Ordinata del centro: --> ");
    double ordinataCentro = it.unibas.utilita.Console.leggiDouble();
    System.out.print("Lunghezza del raggio: --> ");
    double raggio = it.unibas.utilita.Console.leggiDouble();
    Circonferenza circonferenza =
            new Circonferenza(ascissaCentro, ordinataCentro, raggio);
    return circonferenza;
}
```

Compiti dei Componenti

parametro del metodo (riferimento ad una circonferenza)

```
private void schermoStampaDatiCirconferenza(Circonferenza circ) {
    System.out.println("\nDati della circonferenza:");
    System.out.println("Ascissa: " + circ.getAscissaCentro());
    System.out.println("Ordinata: " + circ.getOrdinataCentro());
    System.out.println("Raggio: " + circ.getRaggio());
    System.out.println("Lung. circ: " + circ.getLungCirconferenza());
    System.out.println("Superficie: " + circ.getSuperficieCerchio());
}
```


Compiti dei Componenti

- Gli oggetti di tipo Circonferenza
  - ⇨conoscono le coordinate del proprio centro e la lunghezza del proprio raggio
  - ⇨consentono ad altri componenti di conoscere il valore delle proprietà (metodi “get”) e di cambiarli (metodi “set”)
  - ⇨sanno calcolare la lunghezza della propria circonferenza
  - ⇨e la superficie del proprio cerchio

```
private double ascissaCentro;
private double ordinataCentro;
private double raggio;

public double getAscissaCentro() { ...} ;

public void setAscissaCentro(double ascissaCentro) { ...} ;

public double getOrdinataCentro() { ...} ;

public void setOrdinataCentro(double ordinataCentro) { ...} ;

public double getRaggio() { ...} ;

public void setRaggio(double raggio) { ...} ;

public double getLunghezzaCirconferenza() { ...} ;

public double getSuperficieCerchio() { ...} ;
```

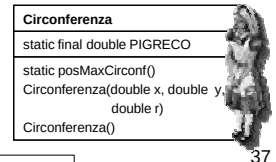

Compiti dei Componenti

- La classe Circonferenza
  - ⇨conosce il valore della costante PIGRECO
  - ⇨è in grado di creare oggetti di tipo Circonferenza (conosce il costruttore e la struttura degli oggetti da creare)
  - ⇨è in grado di effettuare la ricerca di un oggetto attraverso il metodo statico posizioneMassimaCirconferenza()
  - ⇨è una classe di tipologia c.

Compiti dei Componenti

parametro del metodo (riferimento all'array)

```
public static int posMaxCirconf (Circonferenza[] collezione) {
    ...
    int posMass = 0;
    for (int i = 1; i < collezione.length; i++) {
        if (collezione[posMass].getLunghCirconferenza() <
            collezione[i].getLungCirconferenza()) {
            posMass = i;
        }
    }
    return posMass;
    ...
}
```



Compiti dei Componenti

- Le ragioni di questa scelta
  - ⇨corretta attribuzione delle responsabilità
- Responsabilità degli oggetti
  - ⇨conoscere le proprie proprietà
  - ⇨effettuare i calcoli che li riguardano
  - ⇨la ricerca della circonferenza di lunghezza massima non ricade naturalmente tra questi compiti
- Responsabilità naturale per la classe

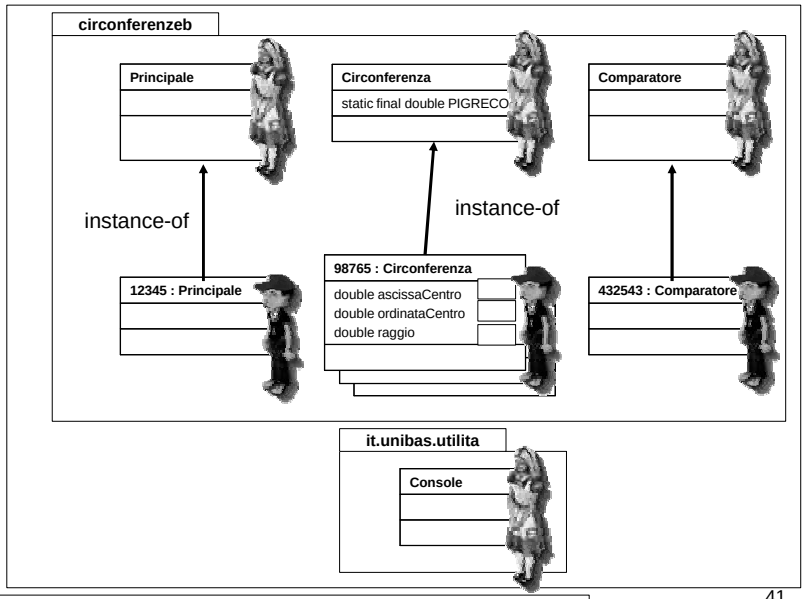
Compiti dei Componenti

- Nota
  - ⇨il metodo statico in questione NON ha effetti collaterali sullo stato del componente
  - ⇨non cambia il valore di nessuna proprietà
  - ⇨riceve parametri e restituisce il risultato
  - ⇨in sostanza, nonostante abbia visibilità globale nell'applicazione, NON ha gli effetti negativi della programmazione con dati globali

Compiti dei Componenti

- Una soluzione alternativa
  - ⇨per cercare la circonferenza più lunga utilizzo un oggetto apposito
  - ⇨oggetto della classe Comparatore
  - ⇨la sua unica responsabilità è quella di confrontare circonferenze e cercare quella più lunga

>> circonferenzeb



Compiti dei Componenti

- Metafora
  - ⇨nella I soluzione la classe si incarica di "supervisionare" il lavoro di tutti gli oggetti che ne sono istanze
  - ⇨e si assume la responsabilità di effettuare elaborazioni relative al loro stato
  - ⇨nella II soluzione viene creato un oggetto "supervisore" appositamente fatto
  - ⇨per analizzare un gruppo di altri oggetti

Esempio

- Svantaggi della II soluzione
  - ⇨maggior numero di componenti e maggiore complessità del codice
- Vantaggi di questa seconda soluzione
  - ⇨non utilizza metodi di visibilità globale
  - ⇨ma si tratta di un vantaggio limitato visto che il metodo statico in questione non ha effetti collaterali

Esempio

- Nel complesso
  - ⇨entrambe le soluzioni sono adeguate
  - ⇨è leggermente preferibile la prima perchè è più semplice
- Linea guida fondamentale della POO

nel caso siano possibili più soluzioni alternative che possono funzionare, nell'incertezza adottare sempre quella più semplice

Riassumendo

- Scegliere il tipo di Componente
- Analisi di Circonferenze
  - ⇨Metodi di Accesso e Modifica
  - ⇨Costruttori con Argomenti
  - ⇨Costanti
  - ⇨Array
- Compiti dei Componenti

Termini della Licenza

- This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

- Questo lavoro viene concesso in uso secondo i termini della licenza “Attribution-ShareAlike” di Creative Commons. Per ottenere una copia della licenza, è possibile visitare http://creativecommons.org/licenses/by-sa/1.0/ oppure inviare una lettera all’indirizzo Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.