Programmazione Orientata agli Oggetti in Linguaggio Java

Sintassi e Semantica
Uso dei Componenti

versione 2.0

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

Sommario

- Componenti
  - ⇨Proprietà
  - ⇨Metodi
  - ⇨Costruttori
- Regole di Visibilità
  - ⇨Modificatori di Visibilità
  - ⇨Visibilità

Componenti

- Due tipi di componenti
  - ⇨classi
  - ⇨oggetti (istanze delle classi)
- Caratteristiche di un componente
  - ⇨proprietà (o “campi” o “attributi”): variabili che descrivono lo stato del componente
  - ⇨metodi: sottoprogrammi che consentono al componente di eseguire le operazioni

Componenti

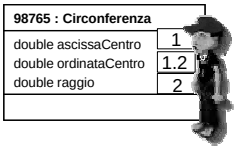
- Messaggi ad un componente
  - ⇨richieste di esecuzione dei metodi
  - ⇨richieste di utilizzo delle proprietà
  - ⇨devono rispettare le regole di visibilità
- Esempi
  - ⇨c.somma(10, 20);
  - ⇨int x = Console.leggiIntero();
  - ⇨int dim = array.length;

Proprietà

ATTENZIONE
una proprietà è una variabile di tipo di base o di tipo riferimento

- Proprietà di un componente
  - ⇨variabile, ovvero spazio nella memoria
  - ⇨allocato nello heap all'interno della memoria riservata al componente
- Due tipi di proprietà
  - ⇨proprietà dei tipi primitivi
  - ⇨proprietà di tipo riferimento

Proprietà



- Proprietà dei tipi primitivi
  - ⇨spazio nella memoria assegnata al componente in cui viene conservato un valore del tipo corrispondente
- Esempio
  - ⇨in Circonferenza:

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

Proprietà

- Proprietà di tipo riferimento
  - ⇨spazio nella memoria assegnata al componente in cui viene conservato un riferimento ad un altro componente
- Esempio
  - ⇨la classe Studente universitario
  - ⇨deve avere le proprietà nome, cognome, matricola, anno di corso

```
package universita;

public class Studente {

    private int matricola;
    private int annoDiCorso;
    private String nome;
    private String cognome;

    public int getMatricola() { return this.matricola; }
    public void setMatricola(int matricola) { this.matricola = matricola; }

    public int getAnnoDiCorso() { return this.annoDiCorso; }
    public void setAnnoDiCorso(int annoDiCorso) { this.annoDiCorso = annoDiCorso; }

    public String getNome() { return this.nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getCognome() { return this.cognome; }
    public void setCognome(String cognome) { this.cognome = cognome; }
}
```

Proprietà

- Per creare uno studente
  - ⇨es: nel metodo main
  - ⇨Studente s = new Studente();
  - ⇨s.setMatricola(1224);
  - ⇨s.setAnnoDiCorso(1);
  - ⇨String nome = new String(“Homer”);
  - ⇨s.setNome(nome);
  - ⇨s.setCognome(“Simpson”);

- creo un oggetto di tipo java.lang.String
- dichiaro un riferimento “nome” all’oggetto creato
- assegno il valore del riferimento alla proprietà omonima dello studente

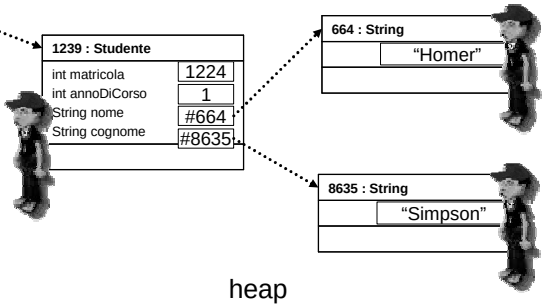
funzionamento analogo (stavolta l’oggetto creato è una stringa costante)

Proprietà

- Nella memoria

#102	...	...
#103	s	#1239
#104	...	

record di attivazione del metodo main



heap

Proprietà

ATTENZIONE
inizializzazione automatica delle proprietà degli oggetti

- Inizializzazione
  - ⇨anche per le proprietà vale la regola di inizializzazione obbligatoria del valore
  - ⇨e anche in questo caso Java aiuta il programmatore
- Inizializzazione automatica delle proprietà
  - ⇨viene fatta automaticamente dalla macchina virtuale alla creazione dell'oggetto
  - ⇨con la "regola del valore nullo"

Proprietà

- Quindi
  - ⇨alla creazione, l'oggetto Studente riceve autom. il valore 0 per matricola e anno di corso e il valore null per nome e cognome
  - ⇨questi valori possono poi essere modificati utilizzando i metodi set
- Nota
  - ⇨conviene sempre inizializzare esplicitamente le proprietà per rendere più leggibile il codice

Proprietà

- Messaggi per utilizzare una proprietà
  - ⇨purchè la proprietà sia visibile all'esterno
- Per le classi
  - ⇨*nomeClasse.proprieta*
  - ⇨es: CalcolatriceStatica.risultato++; // errore! privata
- Per gli oggetti
  - ⇨*riferimento.proprieta*
  - ⇨es: int x = array.length; // se array è un array

Metodi

- Metodi di un componente
  - ⇨sottoprogrammi attraverso i quali il componente esegue le operazioni
  - ⇨possono essere funzioni oppure procedure
- Attenzione alla sintassi per la chiamata
  - ⇨*nomeClasse.nomeMetodo(argomenti);* es: int x = it.unibas.utilita.Console.leggiIntero();
  - ⇨*riferimento.nomeMetodo(argomenti);* es: Calcolatrice c = new Calcolatrice(); c.somma(a, b);

Metodi

- Una stranezza di Java
  - ⇨i membri statici (proprietà e metodi) sono considerati anche membri degli oggetti della classe
  - ⇨e quindi possono essere chiamati, oltre che utilizzando il nome della classe, anche utilizzando un riferimento ad un oggetto della classe
- Ma...
  - ⇨si tratta chiaramente di un'aberrazione
  - ⇨es: in C# non è così

Metodi

- Esempio

```
public class Prova {

   public static void esegui() {
      System.out.println("Prova");
   }

   public static void main(String[] args) {
      Prova p = new Prova();
      p.esegui();
   }
}
```

Metodi

- E' consentito il sovraccarico
  - ⇨purchè i metodi con lo stesso nome siano distinguibili sulla base dei parametri
- Esempi: in java.lang.String
  - ⇨public String substring(int beginIndex);
  - ⇨public String substring(int beginIndex, int endIndex)
- Esempi: in java.lang.Math
  - ⇨public static double abs(double a);
  - ⇨public static int abs(int a);

Metodi

- I dati visibili in un metodo
  - ⇨dati a visibilità globale nell'applicazione (dati public e static); es: costanti, oppure riferimenti come System.out
  - ⇨proprietà del componente che esegue il metodo (globali rispetto ai vari metodi)
  - ⇨parametri
  - ⇨variabili e costanti locali

Metodi

- Attenzione all'inizializzazione
  - ⇨in Java vale la regola di inizializzazione obbligatoria prima dell'uso
  - ⇨le proprietà degli oggetti sono inizializzate automaticamente (regola del valore nullo)
  - ⇨le componenti degli array sono inizializzate automaticamente (regola del valore nullo)
  - ⇨ma le variabili locali ai metodi no

Metodi

- Di conseguenza
  - ⇨le variabili locali dei metodi devono essere inizializzate esplicitamente dal programmatore, o il compilatore si lamenta

```
public static void main (String[] args) {
    int i;
    while (i < 10) {
        System.out.println(i); i++;
    }
}
```

```
E:\tmp\Prova.java:6: variable i might not have
                     been initialized
        while (i < 10) {
```

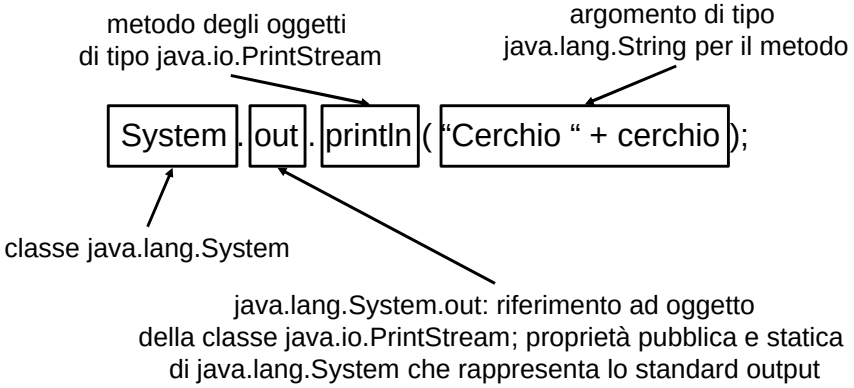
Un Metodo Particolare

- Un metodo particolare
  - ⇨il metodo per la stampa sullo standard output
  - ⇨chiamata del metodo System.out.println
  - ⇨l'oggetto System.out rappresenta lo standard output in un programma Java
- Esempio
  - ⇨System.out.println("Cerchio " + cerchio);

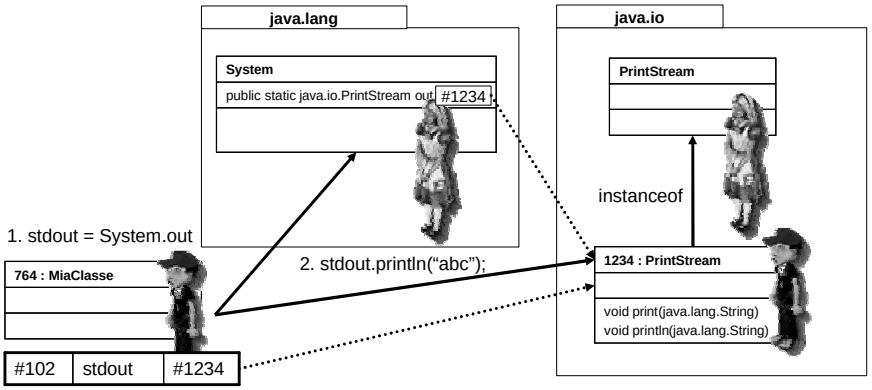
Un Metodo Particolare

equivalente a:
java.io.PrintStream stdout = System.out;
stdout.println("Cerchio"...);

- Anatomia della chiamata



Un Metodo Particolare



System.out.println("abc"); equivalente a:
java.io.PrintStream stdout = System.out;
stdout.println("abc");

Un Metodo Particolare

- In altri termini
  - ⇨nel package java.lang c'è la classe System
  - ⇨tra le proprietà di System c'è
    public static java.io.PrintStream out;
    (proprietà statica e pubblica = var. globale)
  - ⇨tra i metodi di java.io.PrintStream c'è
    public void println(java.lang.String s);
    public void print(java.lang.String s);

Un Metodo Particolare

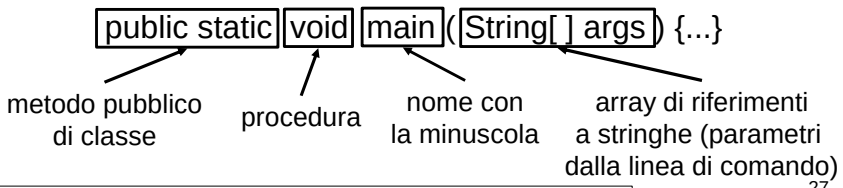
- System.out.println su riferimenti
  - ⇨semantica generale: cerca di trasformare l'oggetto in una sequenza di caratteri da inviare sullo standard output
  - ⇨alcune particolarità
- I particolarità
  - ⇨System.out.println(null) non solleva eccezione, ma stampa i caratteri "null"

Un Metodo Particolare

- II particolarità
  - ⇨per trasformare l'oggetto in una sequenza di caratteri System.out.println chiama automaticamente un metodo toString()
  - ⇨di conseguenza:
    Calcolatrice c = new Calcolatrice();
    System.out.println(c);
  - ⇨equivale a scrivere:
    System.out.println(c.toString());
  - ⇨il metodo toString() è ereditato da Object (>)

Un Altro Metodo Particolare

- Il metodo main
  - ⇨è quello da cui si avvia l'esecuzione di un'applicazione
  - ⇨dev'essere un metodo statico e pubblico
  - ⇨e ha un prototipo stabilito



Un Altro Metodo Particolare

```
package provamain;
public class Prova {

   public static void main(String[] args) {
         if (args.length == 0) {
              System.out.println("Nessun argomento");
         } else {
              for (int i = 0; i < args.length; i++) {
                  System.out.print(args[i] + " ");
              }
         }
   }
}
```

comando: java provamain.Prova
schermo: Nessun argomento

comando: java provamain.Prova primo secondo
schermo: primo secondo

Costruttori

public Circonferenza() {}

- Tutte le classi devono avere almeno un costruttore
  - ⇨se il programmatore non ne sviluppa uno, il compilatore ne aggiunge automaticamente uno standard, senza parametri e vuoto
  - ⇨costruttore “no-arg”
- Attenzione
  - ⇨questo succede SOLO se il programmatore non sviluppa nessun costruttore

Costruttori

- Di conseguenza
  - ⇨se nel codice non viene specificato nessun costruttore, la classe ha il costruttore no-arg
  - ⇨altrimenti avrà i costruttori (uno o più) specificati nel codice
  - ⇨il costruttore è spesso sovraccarico (es: Circonferenza)
  - ⇨il costruttore può essere privato (es: Console)

Regole di Visibilità

- Come tutti i linguaggi
  - ⇨anche Java ha le sue regole di visibilità
- Ma...
  - ⇨le cose sono leggermente complicate dalla presenza dei modificatori di visibilità
  - ⇨che si aggiungono alle regole ordinarie che stabiliscono la visibilità dei dati nei metodi

Modificatori di Visibilità

- Modificatore di visibilità
  - ⇨consentono di stabilire il livello di visibilità di:
  - ⇨una proprietà
  - ⇨un metodo
  - ⇨un'intera classe
- Parole chiave
  - ⇨public
  - ⇨private

Modificatori di Visibilità

- Tre possibilità
- Il modificatore è public
  - ⇨visibilità in tutta l'applicazione
- Il modificatore è private
  - ⇨visibilità solo all'interno della classe e delle sue istanze
- Il modificatore è assente
  - ⇨visibilità "friendly", all'interno del package

Modificatori di Visibilità

- In generale
  - ⇨le proprietà sono private
  - ⇨i metodi sono pubblici o "friendly"
- Attenzione
  - ⇨possono esistere anche proprietà pubbliche (sconsigliato); es: out della classe System
  - ⇨e metodi privati (metodi di servizio della classe, che non devono essere usati all'esterno)

>> circonferenzea.Principale

Modificatori di Visibilità

- Le classi
  - ⇨possono essere esclusivamente public oppure friendly
  - ⇨non ha senso definire classi private (nessuno potrebbe richiedere servizi alla classe)
- In un file .java
  - ⇨può esserci al più una classe pubblica
  - ⇨ma possono esserci varie classi friendly
  - ⇨in generale per ora: un file, una classe

Visibilità

- In un'appl. Java, vari livelli di visibilità
  - ⇨dati locali ad un metodo (parametri, variabili e costanti locali): visibili solo nel metodo
  - ⇨dati locali ad una classe (proprietà private): visibili solo nei metodi della classe
  - ⇨dati locali ad un package: metodi e proprietà "friendly"
  - ⇨dati globali all'applicazione (dati public e static): visibili in tutti i metodi di tutte le classi

dato locale al metodo

costante globale dell'applicazione

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

public double getLunghezzaCirconferenza() {
   return (2 * Circonferenza.PIGRECO * this.raggio);
}

public int getQuadranteCentro() {
   int quadrante = 1;
   if (this.ascissaCentro < 0 && this.ordinataCentro >= 0) {
      quadrante = 2;
   } else if (this.ascissaCentro < 0 && this.ordinataCentro < 0) {
      quadrante = 3;
   } else if (this.ascissaCentro >= 0 && this.ordinataCentro < 0) {
     quadrante = 4;
   }
   return quadrante;
 }
```

98765 : Circonferenza	
double ascissaCentro	3
double ordinataCentro	5
double raggio	6

proprietà dell'oggetto

Visibilità

- Che succede in caso di conflitto di nome?
  - ⇨esempio: dato locale al metodo che si chiama come un dato globale della classe o come un dato globale dell'applicazione
  - ⇨il sistema di nomi di Java consente sempre di risolvere il conflitto
  - ⇨è infatti possibile qualificare i nomi dei dati per distinguerli tra di loro

Visibilità

- Nomi “qualificati”
  - ⇨dati locali al metodo: *<identificatore>* (es: quadrante)
  - ⇨proprietà degli oggetti: this*.<identificatore>* es: this.risultato
  - ⇨proprietà delle classi (locali o globali): *<NomeClasse>.<identificatore>*
    es: CalcolatriceStatica.risultato
    es: Circonferenza.PIGRECO

Visibilità

applicazione per l’analisi di circonferenze

public static final double PIGRECO = 3.14;

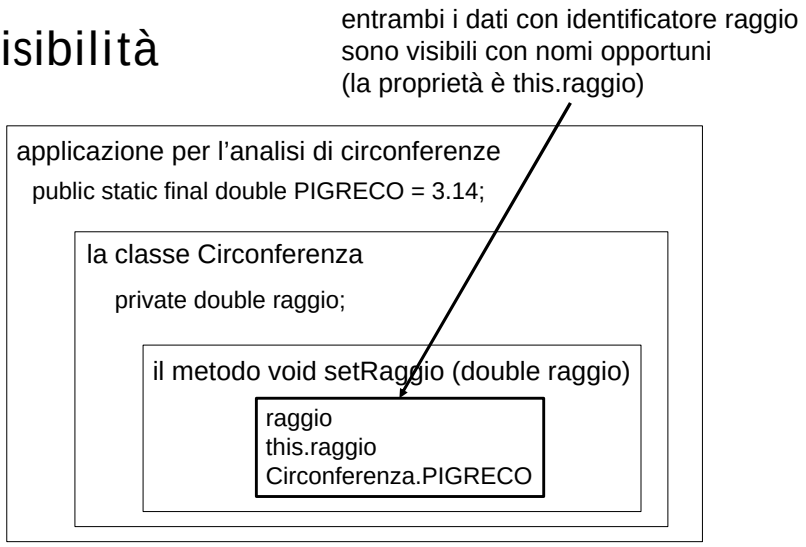
la classe Circonferenza

private double ascissaCentro;

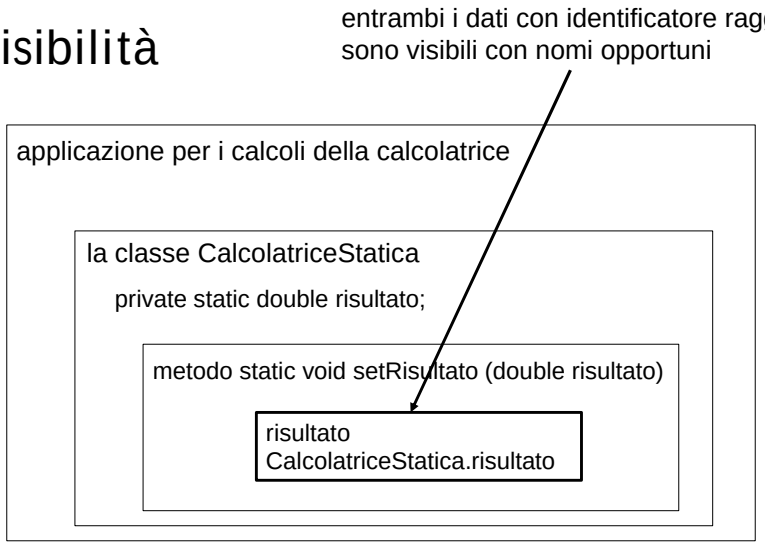
il metodo double getLunghCirconferenza()

int lunghezza;

Visibilità



Visibilità



Visibilità

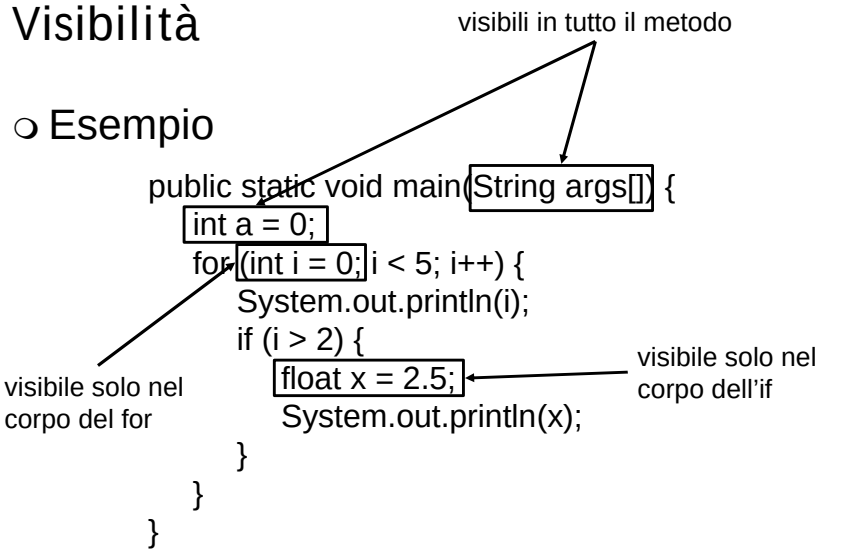
- Nomi abbreviati per le proprietà
  - ⇨per le proprietà di oggetto – se non c'è conflitto con nomi di dati locali – è possibile omettere this (es: raggio e non this.raggio)
  - ⇨per le proprietà di classe – se non c'è conflitto con nomi di dati locali – è possibile omettere il nome della classe (es: risultato e non CalcolatriceStatica.risultato)
  - ⇨ma: è opportuno usare sempre il riferimento completo per migliorare la leggibilità

Visibilità

- Attenzione
  - ⇨in Java valgono le regole di visibilità collegate ai blocchi del C++
- In particolare
  - ⇨una variabile può essere dichiarata in qualsiasi blocco di istruzioni (tra { e })
  - ⇨la variabile è visibile solo all'interno del blocco e non nei blocchi esterni

Visibilità

- Esempio



Riassumendo

- Componenti
  - ⇨Proprietà
  - ⇨Metodi
  - ⇨Costruttori
- Regole di Visibilità
  - ⇨Modificatori di Visibilità
  - ⇨Visibilità

Termini della Licenza