Programmazione in Java
Parte I: Fondamenti

Lezione 3

Dott. Marco Faella

Ereditarietà

- Uno dei principi della programmazione orientata agli oggetti (OOP) è il **riuso**
- Le classi dovrebbero essere progettate come componenti riutilizzabili
- L'ereditarietà consente di riutilizzare una classe esistente, aggiungendovi funzionalità

- L'ereditarietà consiste nel definire una nuova classe A a partire da una data classe B
- La sintassi è la seguente:

class A extends B { ... }

- In questo caso, A viene chiamata *sottoclasse* di B, e B *superclasse* di A
- Gli oggetti di classe A dispongono di tutti i campi e i metodi della classe B
- Tuttavia, per accedere ai membri della classe B, il codice della classe A deve sottostare alle regole di visibilità

- Ad esempio, consideriamo i campi di B...

Ereditarietà dei campi

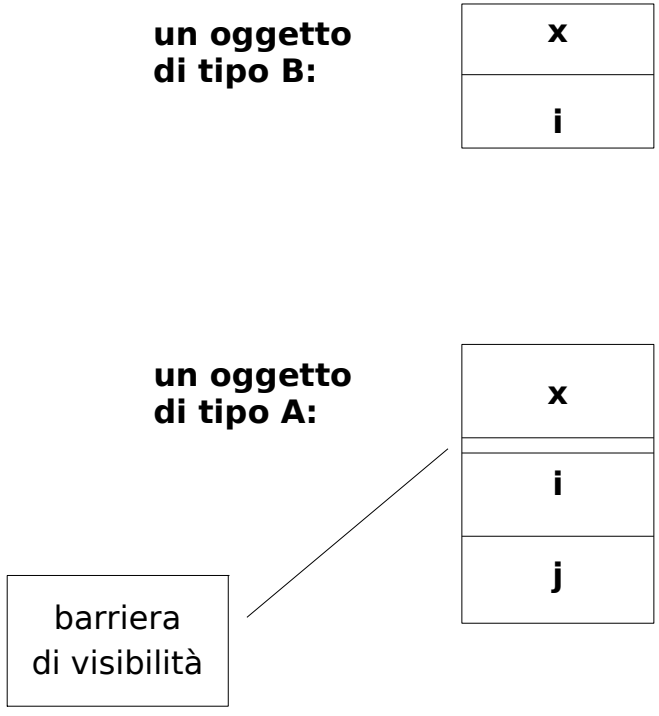
Esempio:

```
class B {
     public int i;
     private double x;
     public double getX() {
          return x;
     }
}

class A extends B {
     public int j;

     public void f() {
          j = 5;
          i = 7;
          x = 3; // errore!
          i = (int)x; // errore!
     }
}
```

Memory layout:



Uso corretto dell'ereditarietà

- Una classe dovrebbe estenderne un'altra solo quando gli oggetti della prima classe sono un tipo particolare di oggetti della seconda
  - In gergo, si dice che la sottoclasse deve essere in relazione “*is a*” con la superclasse

- Ad esempio, supponiamo che Rettangolo sia una classe caratterizzata dagli attributi *base* ed *altezza*
- E' corretto se la classe Quadrato estende Rettangolo, perché un quadrato è un rettangolo i cui lati sono uguali
  - ovvero, un quadrato è un tipo particolare di rettangolo
- Non è corretto se la classe Triangolo estende Rettangolo, perché un triangolo non è un tipo particolare di Rettangolo, *anche se è dotato anch'esso di base e altezza*

Creazione di oggetti della sottoclasse

- Se A estende B, ogni costruttore di A deve necessariamente invocare uno dei costruttori di B
- Un costruttore di A può invocare un costruttore di B usando la parola chiave *super*

```
class A extends B {
     public A(int n) {
          // chiama il costruttore di B che accetta un intero
          super(n);
     }
     public A() {
          // chiama il costruttore di B che accetta due interi
          super(3, 7);
     }
}
```

- L'invocazione super deve essere la prima riga di ciascun costruttore
- Se un costruttore di A non inizia con una chiamata a super, il compilatore inserisce una chiamata del tipo “super()”
- Quindi, viene automaticamente invocato il costruttore di B senza argomenti
- Se un tale costruttore non c'è, viene segnalato un errore di compilazione

Esercizi

- Rettangolo e Quadrato
  - con main separato
- Poligono con getArea e getPerimetro
- Triangolo dati i lati
  - formula di Erone
- Array di Poligoni

Sottotipi

- Un oggetto di tipo A (sottoclasse) può essere assegnato ad un riferimento di tipo B (superclasse)
- Si dice che A è *sottotipo* di B

- Quindi, ogni riferimento ha due tipi:
  - il *tipo dichiarato* (o *statico*)
  - il *tipo effettivo* (o *dinamico*)

- Il tipo dichiarato è quello usato nella dichiarazione del riferimento
  - esso non cambia durante tutta l'esecuzione del programma
- Il tipo effettivo è il tipo dell'oggetto puntato dal riferimento
  - esso può cambiare durante l'esecuzione

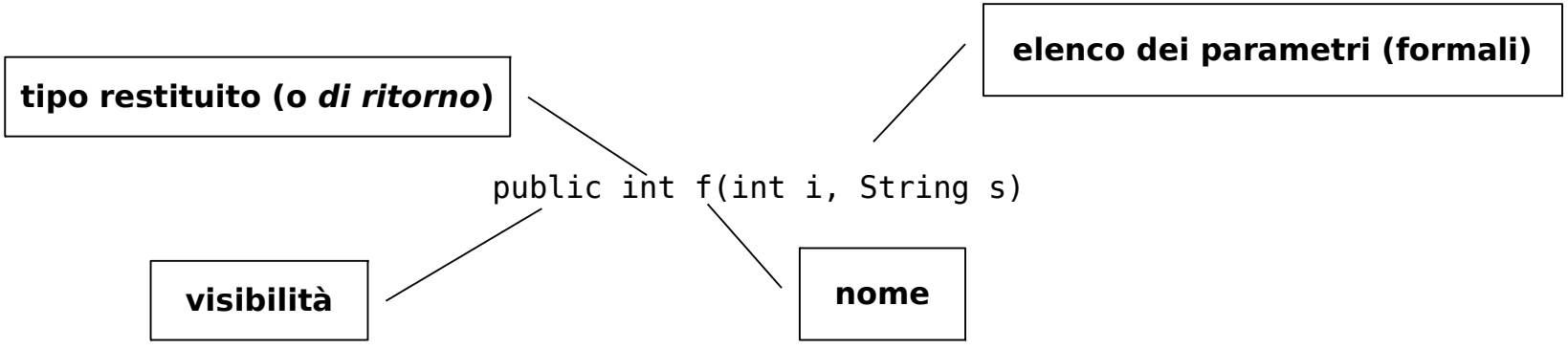
- Ad esempio, nel frammento

```
Poligono p = new Rettangolo(10, 5);
```

- diremo che:
  - il tipo dichiarato di “p” è Poligono (e tale resterà per tutto il programma)
  - il tipo effettivo di “p”, in questo momento, è Rettangolo (ma potrebbe cambiare in seguito ad una nuova assegnazione)

L'overriding

- Una sottoclasse può *ridefinire* un metodo della superclasse, ovvero fornirne una versione più specifica
- Questa possibilità prende il nome di *overriding*
- Dato un metodo nella superclasse, ad es.:



- La sottoclasse può a sua volta definire un metodo con lo stesso nome, stessi parametri, stesso tipo di ritorno* e stessa visibilità*
- Invocando un metodo su un riferimento, verrà invocata **la versione più specifica** di quel metodo, cioè quella corrispondente al **tipo effettivo** del riferimento

- (*) Nota: il tipo di ritorno e la visibilità possono anche essere diversi, a certe condizioni