Java: Classi interne

Marco Faella

Dip. Ing. Elettrica e Tecnologie dell'Informazione
Università di Napoli “Federico II”

Corso di Linguaggi di Programmazione I

Classi interne

- Java permette di definire una classe (o interfaccia) all'interno di un'altra
- Queste classi vengono chiamate “interne” o “annidate”
  - in inglese, il termine usato è *nested*
  - In particolare, le classi interne non statiche sono chiamate *inner*

- Tale meccanismo arricchisce le possibilità di relazioni tra classi, introducendo in particolare **nuove regole di visibilità**

Proprietà delle classi interne

Le classi interne (non statiche) godono delle seguenti proprietà distintive:

1) **Privilegi di visibilità** rispetto alla classe contenitrice e alle altre classi in essa contenute
   - permettono una stretta collaborazione tra queste classi

2) **Restrizioni di visibilità** rispetto alle classi esterne a quella contenitrice
   - permettono di nascondere la classe all'esterno (incapsulamento)

3) Un **riferimento implicito** ad un oggetto della classe contenitrice
   - ogni oggetto della classe interna "conosce" l'oggetto della classe contenitrice che l'ha creato

Restrizioni di visibilità delle classi interne

- Oltre a campi e metodi, una classe può contenere altre classi o interfacce, dette “interne”
- Una classe che non sia interna viene chiamata “top-level”

- A differenza delle classi top-level, le classi interne possono avere **tutte le quattro visibilità** ammesse dal linguaggio
- La visibilità di una classe interna X stabilisce quali classi possono utilizzarla (cioè, istanziarla, estenderla, dichiarare riferimenti o parametri di tipo X, etc.)

Restrizioni di visibilità delle classi interne

Consideriamo il seguente esempio:

```
public class A {
     private class B {
          ...
     }
     class C {
          ...
     }
}
```

La classe B **non è visibile al di fuori di A**

La classe C è visibile a tutte le classi che si trovano nello stesso pacchetto di A

Privilegi di visibilità delle classi interne

```
public class A {
   private class B {
      ...
   }
   class C {
      ...
   }
}
```

- Dall'esterno di A, i nomi completi delle classi B e C sono A.B e A.C, rispettivamente
- La visibilità di una classe interna non ha alcun effetto sul codice che si trova all'interno della classe che la contiene
- Ad esempio, la classe B dell'esempio è visibile a tutto il codice contenuto in A, compreso il codice contenuto in altre classi interne ad A, come ad esempio C
- Lo stesso discorso si applica per i campi e i metodi di una classe interna
  - i loro attributi di visibilità hanno effetto solo sul codice *esterno* alla classe contenitrice
- In altre parole, **tra classi contenute nella stessa classe non vige alcuna restrizione di visibilità**

Relazione tra oggetti

- Ciascun oggetto di una classe interna (non statica, come spiegato dopo) possiede un **riferimento implicito** ad un oggetto della classe contenitrice
- Tale riferimento viene inizializzato automaticamente al momento della creazione dell'oggetto
- Tale riferimento non può essere modificato

- Supponiamo che B sia una classe interna di A
- All'interno della classe B, la sintassi per denotare questo riferimento implicito è **A.this**

- L'uso di A.this è facoltativo, come quello di this
  - cioè, B può accedere ad un campo o metodo “f” della classe A sia con “A.this.f” che con “f”

Uso del riferimento implicito

Esempio:

```
public class A {
     private int x;

     public class B {
          private int y;
          public void stampami() {
               System.out.println(A.this.x);    // uso del riferimento implicito
               System.out.println(y);
          }
     }
}
```

Contesti statici e non

- Il **contesto statico** di una classe è quella porzione del codice di quella classe che si trova nell'ambito di validità di un modificatore static
- Quindi, in una delle seguenti posizioni:
  - All'interno di un metodo statico
  - All'interno di un blocco di inizializzazione statico
  - Nella definizione di un attributo statico

- La parte restante della classe si chiama **contesto non statico**

- Supponiamo che B sia una classe interna di A
- Se viene creato un oggetto di tipo B in un *contesto non statico* della classe A, il riferimento implicito verrà inizializzato con il valore corrente di this
- In tutti gli altri casi, è necessario utilizzare una **nuova forma dell'operatore "new"**, ovvero:

<riferimento ad oggetto di tipo A>.new B(...)

Istanziare una classe interna

Esempio: creazione di un oggetto di classe interna in un *contesto non statico* della classe esterna

```
public class A {
     private int x;

     public class B {
          private int y;
          public void stampami() {
               System.out.println(A.this.x);   // uso del riferimento implicito
               System.out.println(y);
          }
     }

     public B makeB(int val) {
          B b = new B();     // il nuovo oggetto B è legato a this
          b.y = val;         // privilegi di visibilità
          return b;
     }
}
```

Istanziare una classe interna

Esempio: creazione di un oggetto di classe interna in un contesto *statico* della classe esterna

```
public class A {
     private int x;

     public class B {
          private int y;
          public void stampami() {
               System.out.println(A.this.x);   // uso del riferimento implicito
               System.out.println(y);
          }
     }

     public static void main(String[] args) {
          B b = new B();     // errore di compilazione
          A a = new A();
          B b = a.new B();   // OK
     }
}
```

Classi interne statiche

Classi interne statiche

- Le classi interne possono essere statiche o meno
- Una classe interna dichiarata nello scope di classe (cioè al di fuori di metodi e inizializzatori) è statica se preceduta dal modificatore “static”
- Le classi interne possono anche trovarsi all'interno di un metodo (parleremo di classi *locali*), ma non ne parleremo in questo corso
- **Le classi interne statiche non possiedono il riferimento implicito** alla classe contenitrice
- Nota: prima di Java 16, una classe interna non statica non poteva avere membri (campi o metodi) statici

Proprietà delle classi interne

- Riassumendo, le classi interne non statiche godono delle seguenti proprietà distintive:

  1) **Privilegi di visibilità** rispetto alla classe contenitrice e alle altre classi in essa contenute
     - permettono una stretta collaborazione tra queste classi

  2) **Restrizioni di visibilità** rispetto alle classi esterne a quella contenitrice
     - permettono di nascondere la classe all'esterno (incapsulamento)

  3) Un **riferimento implicito** ad un oggetto della classe contenitrice
     - ogni oggetto della classe interna "conosce" l'oggetto della classe contenitrice che l'ha creato

- Le classi interne *statiche* godono solo delle **prime due** proprietà

Nota: alcuni testi si riferiscono a tutte le classi interne come "annidate" e riservano il termine "interne" solo alle classi interne non statiche

Privilegi di visibilità delle classi interne

Il seguente esempio mostra come le classi interne (anche statiche) abbiano pieno accesso ai membri privati della classe contenitrice

```
public class External
{
   private int n = 42;

   public static class A {
      public void foo(External guest) {
         // This is legal
         guest.n = 0;
     }
   }
   public static class B extends External {
      public void foo() {
         // Wrong syntax (compile-time error)
         n = 0;
         // Wrong syntax (compile-time error)
         this.n = 0;
         // This is legal
         super.n = 0;
     }
   }
}
```

Se le classi A e B fossero top-level, non potrebbero accedere al campo *n*.

Memory layout di classi interne

- La figura rappresenta il memory layout delle classi definite di seguito
- Notate in particolare la differenza tra classe interna e sottoclasse

```
public class A {
    private int x;
    private class B {
            private int y;
    }
    static class C {
            private double z;
    }
    public static class D extends A {
            private int w;
    }
    public static void main(String[] args) {
            A a = new A();
            B b = a.new B();
            C c = new C();
            A d = new D();
    }
}
```

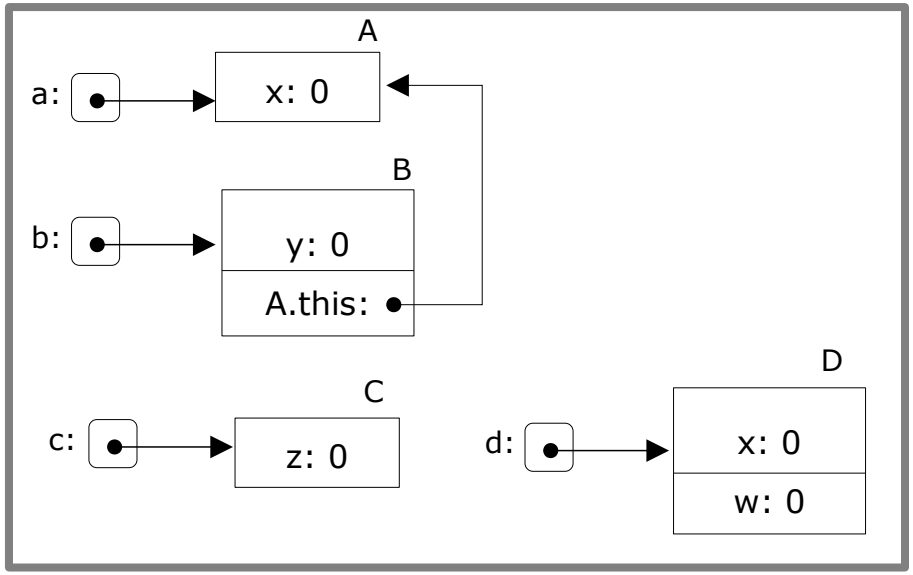

Figura 1: Il memory layout del frammento di programma a sinistra.

- **C++**
  - [*Come Java*] Prevede classi interne (nested) con restrizioni di visibilità arbitraria (ad es., private)
  - [*Come Java*] La classe interna ha accesso a tutti i membri della contenitrice
  - La classe esterna *non* ha accesso privilegiato al contenuto di quella interna
  - Non prevede riferimento implicito alla classe esterna
  - Privilegi di visibilità si ottengono anche tramite il costrutto *friend*

- **C#**
  - Regole analoghe a C++
  - Nessun costrutto friend
  - Le linee guida MS sconsigliano esplicitamente di creare classi interne pubbliche

Esercizio

Nell'ambito di un programma di geometria, si implementi la classe Triangolo, il cui costruttore accetta le misure dei tre lati. Se tali misure non danno luogo ad un triangolo, il costruttore deve lanciare un'eccezione. Il metodo getArea restituisce l'area di questo triangolo.

Si implementino anche la classe Triangolo.Rettangolo, il cui costruttore accetta le misure dei due cateti, e la classe Triangolo.Isoscele, il cui costruttore accetta le misure della base e di uno degli altri lati.

Si ricordi che:

- Tre numeri a, b e c possono essere i lati di un triangolo a patto che a<b+c, b<a+c e c<a+b.
- L'area di un triangolo di lati a, b e c è data da: $\sqrt{p(p-a)(p-b)(p-c)}$ (formula di Erone), dove p è il semiperimetro.

Esempio d'uso (fuori dalla classe Triangolo):

```
Triangolo x = new Triangolo(10,20,25);
Triangolo y = new Triangolo.Rettangolo(5,8);
Triangolo z = new Triangolo.Isoscele(6,5);

System.out.println(x.getArea());
System.out.println(y.getArea());
System.out.println(z.getArea());
```

Output dell'esempio d'uso:

```
94.9918
19.9999
12.0
```