Nested e Inner class

di Roberta Molinari

Nested e Inner class

Finora abbiamo utilizzato solo classi **top level** ovvero ad ogni classe corrisponde un file dedicato. È possibile dichiarare dentro delle **outer class** (*esterne*) altre classi: le **nested class** (*annidate* o *interne*). Possono essere:

1. static member classes, enums e interfaces
2. non-static member classes
3. local classes
4. anonymous classes

Nei casi 2,3,e 4 sono dette **inner class**.

- Generalmente sono **private**
- Al momento della compilazione si creerà un file .class per ogni classe nested dichiarata all'interno dello stesso file .java

Nested e Inner class

```
class Esempio {                          //outer class
  static class SMC {/*...*/}   //1 Static member class NESTED
  interface SMI {/*...*/}      //1 Static member interface NESTED
  enum SME {/*...*/}           //1 Static member enum NESTED

  class NSMC {/*...*/}         //2 Non-static member(INNER)class

  void nsm() {          //metodo classe top
     class NSLC {/*...*/}     //3 Local(INNER) in NoStatCont
  }
  static void sm() {  //metodo statico classe top
     class SLC {/*...*/}      //3 Local(INNER) in StatCont
  }

  SMC nsf = new SMC() {        //4 Anonymous(INNER) in NoStatCont
                /*...*/
  }; ←notare il ; perché è un'istruzione unica, è un "attributo"
  static SMI sf = new SMI() { //4 Anonymous (INNER) in StatCont
                /*...*/
  }; ←notare il ;
//le anonymous possono essere: attributi, locali o parametri
}
```

1
2
3
4

Nested e Inner class
Perché usarle

- È un modo di raggruppare logicamente classi che vengono utilizzate solo in quell'unico contesto: se una classe è utile per una sola altra classe, allora è logico incorporare una nell'altra e mantenere le due classi insieme.
- Aumentano l'incapsulamento: se ho due classi di primo livello, A e B, ma B ha bisogno di accedere ai membri privati di A, nascondendo B nella classe A, i membri di A possono rimanere privati e B può accedervi. Inoltre, B stesso può essere nascosto dal mondo esterno.

Nested e Inner class
Perché usarle

- Rende il codice più leggibile e gestibile: piccole classi nested all'interno di classi di primo livello pone il codice più vicino a dove viene utilizzato.
- Implementano la **composizione forte**

Nested e Inner class
Visibilità

Le classi interne possono avere tutte le quattro visibilità ammesse dal linguaggio.

```
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.

La classe B non è visibile al di fuori di A, mentre la classe C è visibile a tutte le classi che si trovano nello stesso package di A. B è visibile a tutto il codice contenuto in A, compreso il codice contenuto in altre classi interne ad A, come ad esempio C.

Nested e Inner class

1.Nested class statiche

- In java non è possibile avere una classe statica di primo livello.
- Le classi **nested static member class** si comportano come i **membri statici:** non c'è bisogno di istanziare prima la top class, in quanto non contengono nessun riferimento ad esse.
- Sono accessibili dall'esterno usando

```
new TopClass.NestedClass()
```

- In pratica si comportano come classi top level inserite in classi top level

Nested e Inner class

1.Nested class statiche

- Una differenza con le inner class, è che queste non hanno accesso ai membri della classe top level in quanto come appena detto non contengono nessun riferimento ad esse (come i metodi statici non hanno accesso alle variabili di istanza e ai metodi non-static di una classe.)
- Possono accedere solo ai membri statici della classe top level

Nested e Inner class

1.Nested class statiche

```
class A{
    static class ClasseAnnidata1{
       static void stampaQualcosa(){
        System.out.println("metodo statico classe annidata");
       }
    }
}
class EsempioNested{
    private int var2=25;
    static int VAR2=20;
    static class ClasseAnnidata2{
            void  stampaQualcosa2(){
            //var2=23;    non può accedere non è static
            VAR2++;
            System.out.println("stampa annidata VAR2="+ VAR2);       }
    }
}
class TestNested{
    public static void main(String[] args)    {
        A.ClasseAnnidata1.stampaQualcosa();//metodo statico
        EsempioNested.ClasseAnnidata2  b2 =
                                        new EsempioNested.ClasseAnnidata2();
        b2.stampaQualcosa2();
    }
}
```

Nested e Inner class

2.Non-static member

Non-static member class: inner class definita senza la keyword *static*.

- Una non-static member class NON può avere membri static (a meno che non siano anche *final*)
- Le inner class hanno accesso diretto a ogni membro (inclusi i nested) anche se dichiarato *private* della outer class. Si può scrivere
  - `nomeAttributo`
  - **`OuterClass.this`**`.nomeAttributo`
- Nella sua dichiarazione si possono usare:
  - **`final`**
  - **`abstract`**
  - **`public, private, protected`**

Nested e Inner class

2.Non-static member

- Un'istanza di una non-static member class può esistere solo e soltanto insieme all'istanza della classe che la contiene (si deve prima istanziare un oggetto della outer classe e poi istanziare un oggetto della inner class)
  - Dentro un metodo della outer class

```
InnerClass i= new InnerClass();
```

  - In un metodo della classe client della outer

```
TopClass t= new TopClass();
TopClass.InnerClass i= t.new InnerClass();
```

- La creazione dell'istanza deve avvenire all'interno di una istanza di una Outer Class.
- Se si prova a creare un oggetto di tipo Inner Class all'interno di un metodo statico si otterrà un errore di compilazione.

Nested e Inner class

2.Non-static member

```
class ClasseEsterna { //Outer class
    private int x = 5;
    public  void creaInnerClass() {
        ClasseInterna in = new ClasseInterna ();
        in.stampaX();
    }
     class ClasseInterna {
        public void stampaX() {
            System.out.println("x=" + x);
            System.out.println("Riferimento Inner " +
                                this);
            System.out.println("Riferimento Outer " +
                        ClasseEsterna.this);
         }
     }
}
```

Nested e Inner class

2.Non-static member

```
public class Casa {
    private String nome;
    public Casa(String nome) { this.nome = nome; }
    public String getNome() {
        return("Questa casa è di:" + nome); }

    private class Stanze {
      private int numeroStanze=2;
      public String getStanzeENome() {
         return Casa.this.nome + " ha una casa con " +
                        numeroStanze +" stanze.";
      }
      public int getStanze(){ return this.numeroStanze;}
   }
}
```

Nested e Inner class

2.Non-static member

```
public static void main(String[] args) {
    Casa home = new Casa("Federico");
    Casa.Stanze room = home.new Stanze();
    System.out.println(home.getNome());
     //Questa casa è di:Federico
    System.out.println(room.getStanzeENome());
     // Federico ha una casa con 2 stanze.
    }
}
```

Nested e Inner class
Situazione in memoria

- Ciascun oggetto di una classe interna (non statica) 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 interna B, la sintassi per denotare questo riferimento implicito è A.this

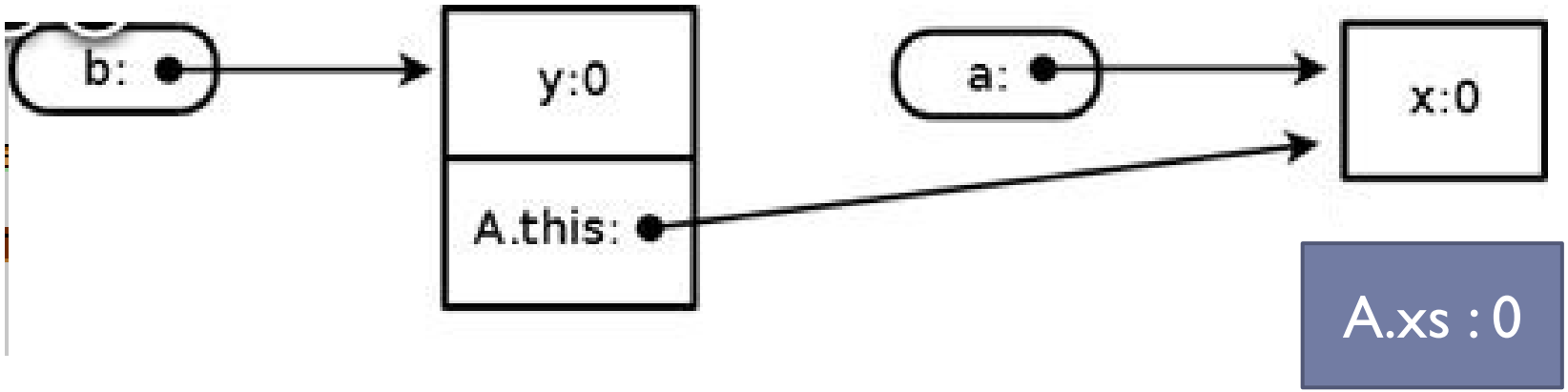
- Nei contesti statici di B, siccome non è disponibile this, non è disponibile neanche A.this.
- L'uso di A.this è facoltativo, come quello di this, cioè, si può accedere ai campi e ai metodi della classe A anche direttamente

Nested e Inner class
Situazione in memoria

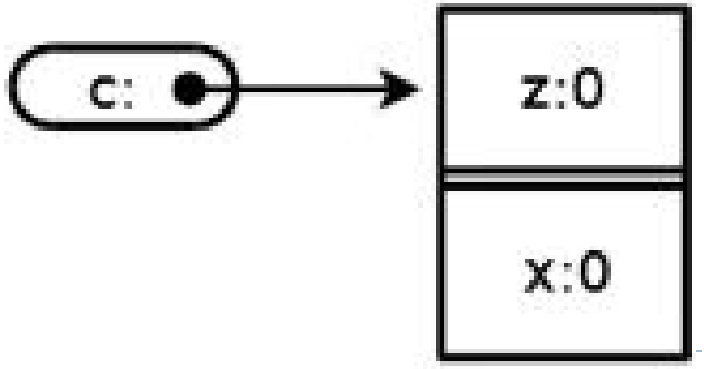
```
class A {
    private int x;
    private static int xs
    public void metA(){
        B b = new B();
        C c = new C();
    }
    public class B {private int y=x;}

    public static class C extends A {private int z=sx;}
}

public class EsNonStaticMember{
    public static void main(String[] args) {
        A.C c = new A.C();
        A a = new A();
        a.metA();
        A.B b = a.new B();
    }
}
```

Nested e Inner class
Situazione in memoria

- Gli oggetti della classe interna hanno un riferimento alla classe esterna (Outer.this)



- Le classi statiche non hanno riferimento all'outer class



Nested e Inner class

3.Local class

Local class (classe locale): inner class che è definita in un blocco di codice (il corpo di un metodo, di un costruttore, un blocco locale, un inizializzatore statico o un inizializzatore di istanza). Ha le seguenti caratteristiche, simili alle variabili locali:

- Non può avere membri statici (questo però non esclude la possibilità di avere un membro *final static*, essendo esso una costante)
- Non può avere modificatori di accessibilità (public, private, ecc.) perché segue le regole di una variabile locale. Sarà quindi dichiarata scrivendo semplicemente *class nomeClasse {…}*
- Può essere *abstract o final*
- Può estendere un'altra classe
- Può essere istanziata solo nel blocco nel quale è definita e deve essere dichiarata prima di essere utilizzata
- Sarà statica o meno a seconda del tipo di metodo in cui è dichiarata

Nested e Inner class

3.Local class

- All'interno del blocco in cui è definita, una local class può accedere solo alle variabili locali dichiarate *final* o effettivamente final. Questo perché il riferimento alla local class vive nella Heap, mentre le variabili locali vivono nella stack solo per il tempo di esecuzione del metodo
- Una non-static local class può accedere sia ai membri statici sia ai membri non statici della classe contenitore (da non confondere con il blocco contenitore) anche privati
- Una static local class può accedere solo ai membri statici della classe che la contiene, perché non ci sono istanze collegate
- Se il blocco contenente la dichiarazione della local class è definito in un contesto statico (ad esempio un metodo statico o un inizializzatore statico) allora la local class è implicitamente statica e quindi non richiede alcun oggetto contenitore per essere istanziata. Ma anche in questo caso non è consentito utilizzare nella propria dichiarazione la parola chiave *static*.

Nested e Inner class

3.Local class

```
public class OuterClass {   // Classe Esterna
  private String x = null;
  void metodoDiTest(String s)   {
    int w=0;    //effettivamente final
    final String y=null;
    String z = null;
    class ClasseInterna {//NON static
      public void stampaX(String s){
          System.out.println("x=" + x +" s="+s);
          System.out.println("Var local w=" + w+ " y=" + y);
          System.out.println("Var local z=" + z); //ERRORE
     } }
    ClasseInterna in= new ClasseInterna();
    z = "variabile locale";  //modificata effetivamente NON final
    in.stampaX("parametro");
   }
   public static void main(String[] args) {
       OuterClass es= new OuterClass ();
       es.metodoDiTest();   }}
}
```

perché il metodo in cui non è final

Nested e Inner class

3.Local class

```
public class OuterClass {   // Classe Esterna
  private String x = null;
  int a=0;
  static void metodoDiTest()   {
    int w=0;    //effettivamente final
    final String y=null;
    class ClasseInterna {//implicitamente static
      public void stampaX(String s){
          System.out.println("x=" + x +" s="+s);
          System.out.println("a=" + a); //ERRORE
          System.out.println("Var local w=" + w+ " y=" + y);
     } }
    ClasseInterna in= new ClasseInterna();
    z = "variabile locale";
    in.stampaX("parametro");
   }
   public static void main(String[] args) {
       OuterClass es= new OuterClass ();
       es.metodoDiTest();   }}
}
```

Nested e Inner class

4.Anonymous class

Anonymous class (classe anonima): inner class che viene contemporaneamente definita e istanziata e non ha un nome.

- La sintassi di dichiarazione e implementazione è

```
<superClasse> x= new <superClasse> (<lista
 opzionale di argomenti>) {…};
<interfaccia> i= new <interfaccia> () {…};
```

Questo è il solo caso in cui si può assistere alla parola chiave **new** affiancata dal nome di una interfaccia in quanto non si sta istanziando un oggetto, ma si sta creando una istanza di una classe anonima che implementa un'interfaccia

- Anche se una classe anonima estende una classe o implementa un'interfaccia (solo una!), non usa né la clausola extends né la clausola implements.

Nested e Inner class

4.Anonymous class

- Esiste se e soltanto se esiste una super classe da estendere o un'interfaccia da implementare!
- È utile quando una classe interna viene utilizzata soltanto una volta, per es. per istanziare un oggetto che poi viene mascherato con una classe o interfaccia nota.
- Può essere dichiarata come "attributo", come local class o come parametro
- Dovrebbero essere utilizzate solo se costituite da poche righe di codice: il suo utilizzo è guidato dalla necessità di ridefinire un metodo di una superclasse in maniera molto veloce.
- Si usano nelle applicazioni grafiche

Nested e Inner class

4.Anonymous class

- Possono essere dichiarati degli attributi, ma non static (non c'è il nome della classe).
- Come per le local classes, anche le anomymous classes non possono usare la parola chiave static in fase di dichiarazione.
- Vedono tutti gli attributi della classe
- Possono usare le variabili del metodo in cui sono dichiarate, ma solo quelle non modificabili (le costanti final).
- Da Java 8 se una var assume un solo valore nel programma la considera come costante final (resta sottinteso), perciò si può omettere il final.

Nested e Inner class

4.Anonymous class

Esempio con estensione di **classe**

```
class A{
   public void stampa(){System.out.println("A");}
}
class B{
   A a = new A() {
         public void stampa()  {
               System.out.println("anonymous A");}
        }; //notare il ;
   a.stampa(); //anonymous A
}
```

Sono state definite due classi: A e B. La prima ha un metodo chiamato stampa() e la seconda ha una variabile di istanza di tipo A. La variabile a non si riferisce ad una istanza di A, ma a una istanza di una sottoclasse Anonima di A che ridefinisce il metodo stampa().

Nested e Inner class
4.Anonymous class

Esempio con implementazione di **interface**

```
interface A{
   public void stampa();
}

class B{
   A a = new A() {
            public void stampa(){
                System.out.println("implementer anonimo");
            }
        };
   a.stampa(); // implementer anonimo
}
```

Nested e Inner class

4.Anonymous class

Esempio come **parametro**

```
class A{
  void metodoDiTest() {
        B b = new B();
        b.do( new In(){
              public void faiQualcosa(){
                    System.out.println("fatto");
              } });
  }
}
interface In{ void faiQualcosa();}
class B{
   void do(In f) {
       f.faiQualcosa();} //non ancora implementato
}
```