Programmazione Orientata agli Oggetti in Linguaggio Java

Tecniche di Programmazione: Classi Interne

versione 1.2

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

Sommario

- Introduzione
- Classi Nidificate
- Classi Interne
- Implementazione
- Classi Interne Anonime

Introduzione

- Nei linguaggi basati sulla riflessione
  - una classe per la macchina virtuale è un oggetto (java.lang.Class)
  - un metodo è un oggetto di tipo java.lang.reflect.Method, ed un blocco di codice oggetto (analogamente un costruttore)
  - una proprietà è un oggetto di tipo java.lang.reflect.Field, oltre che uno spazio nello heap

Introduzione

- L'oggetto di tipo Class
  - mantiene riferimenti agli oggetti che rappresentano i membri di una classe (proprietà, metodi, costruttori)
- Una caratteristica interessante
  - questo meccanismo consente alle classi di avere tra i propri membri altre classi
  - riferimento tra l'oggetto class della classe contenitore e quello della classe contenuta

Introduzione

- Membri di una classe
  - ⇨proprietà (statiche e non)
  - ⇨metodi (statici e non)
  - ⇨costruttori (esclusivamente statici)
  - ⇨altre classi (statiche e non)
- Terminologia per le classi membro
  - ⇨classe membro statica = classe nidificata
  - ⇨classe membro non statica = classe interna

Introduzione

- Un esempio

classe ordinaria
(classe “esterna”)
prova.Esterna

classe membro
(classe interna)
prova.Esterna.Interna

```
package prova;
public class Esterna {

    public static final int COSTANTE = 0;
    private int attributoA;

    public void metodoB() {...}

    private class Interna {

        private String attributoB;

        public int metodoB() {
            ...
        }

    }
}
```

Introduzione

- Le classi membro
  - ⇨hanno le caratteristiche degli altri membri
  - ⇨possono essere pubbliche, private, protette o friendly
  - ⇨il loro nome è *ClasseEsterna.ClasseInterna*
  - ⇨vengono ereditate dalle sottoclassi
- Nota sulla visibilità
  - ⇨è una differenza rispetto alle classi ordinarie che possono essere solo pubbliche o friendly

Introduzione

- Le due caratteristiche delle classi membro
  - ⇨tipicamente la classe membro è privata, ed è quindi visibile esclusivamente nell'ambito del codice della classe contenitore
  - ⇨essendo parte dell'implementazione (codice) della classe contenitore, nella classe membro sono accessibili tutti gli altri membri della classe contenitore, inclusi quelli privati

Introduzione

- Perchè utilizzare una classe membro ?
  - ⇨tre applicazioni fondamentali
- I ragione: privatezza
  - ⇨la classe membro genera componenti che vengono utilizzati esclusivamente all'interno del codice della classe contenitore e non è necessario che siano visibili all'esterno
  - ⇨tipica applicazione delle classi nidificate

Introduzione

- II ragione: legame stretto tra oggetti
  - ⇨la classe membro genera componenti devono lavorare a stretto contatto con i componenti della classe contenitore
  - ⇨accedendo alla loro implementazione e non solo all'interfaccia
  - ⇨tipica applicazione delle classi interne

Introduzione

- III ragione: ereditarietà multipla
  - ⇨nel caso in cui una classe deve ereditare l'implementazione di più classi astratte o concrete può estenderne direttamente una
  - ⇨e poi definire classi membro che estendono le altre per ereditare indirettamente anche da queste
  - ⇨il problema non sorge nel caso dell'ereditarietà multipla di tipo

Introduzione

ATTENZIONE all'utilizzo delle classi membro

- Attenzione alle classi membro
  - ⇨si tratta di uno strumento di programmazione avanzato
  - ⇨che prevede moltissimi dettagli di carattere sintattico e semantico
- Linea guida
  - ⇨vanno utilizzati solo nei casi tipici visti
  - ⇨e solo se non è possibile utilizzare altre soluzioni efficaci

Classi Nidificate

- Classe nidificata
  - ⇨classe membro statica
- Utilizzo tipico
  - ⇨la classe nidificata è privata
  - ⇨viene utilizzata esclusivamente all'interno della classe contenitore come meccanismo interno di organizzazione del codice

Classi Nidificate

- Un esempio
  - ⇨la classe java.util.LinkedList
- La classe nidificata LinkedList.Entry
  - ⇨classe privata i cui oggetti rappresentareno gli elementi della lista collegata
  - ⇨valore, riferimento al precedente, riferimento al successivo

```
package java.util;

public class LinkedList extends AbstractSequentialList
                        implements List, Cloneable, java.io.Serializable {

  // classe interna statica Entry
  private static class Entry {
     Object element;
     Entry next;
     Entry previous;

     Entry(Object element, Entry next, Entry previous) {
        this.element = element;
        this.next = next;
        this.previous = previous;
     }
  }

  // proprietà
  private Entry header = new Entry(null, null, null);
  private int size = 0;

  // costruttori
  public LinkedList() {
     header.next = header.previous = header;
  }
  ...
```

codice della classe nidificata LinkedList.Entry

la classe contenitore manipola le proprietà degli oggetti della classe annidata direttamente

Classi Nidificate

- Vantaggi dell'utilizzo della classe membro
  - ⇨aiuta ad organizzare il codice di LinkedList
  - ⇨senza esporne all'esterno l'implementazione (può essere facilmente cambiata senza intaccare il resto dell'applicazione)
  - ⇨nei metodi di LinkedList è possibile accedere direttamente alle proprietà di Entry
  - ⇨anche se fossero dichiarate private

Classi Nidificate

- Il reale significato di private
  - ⇨“visibile all’interno del codice della classe a cui il membro privato appartiene”
- Ovvero
  - ⇨visibile nel codice della classe
  - ⇨e in quello di tutte le sue classi membro

Classi Interne

- Classe interna
  - ⇨classe membro non statica
- Differenza rispetto alla classe nidificata
  - ⇨la classe nidificata è collegata ad un componente di tipo classe (classe esterna)
  - ⇨la classe interna è invece un membro degli oggetti della classe contenitore, per cui il legame è tra oggetti della classe interna e oggetti della classe contenitore

Classi Interne

ATTENZIONE
all'utilizzo tipico
delle classi interne

- Utilizzo tipico
  - ⇨la classe interna è privata
  - ⇨ma implementa un'interfaccia pubblica
  - ⇨gli oggetti della classe interna sono usati all'esterno attraverso riferimenti all'interfaccia
  - ⇨vengono costruiti e forniti all'esterno attraverso metodi della classe contenitore
  - ⇨collaborano strettamente con gli oggetti della classe esterna

Classi Interne

>> java.util.LinkedList

- Un esempio
  - ⇨sempre basato su java.util.LinkedList
- La classe LinkedList.ListItr
  - ⇨realizza un iteratore per la LinkedList
  - ⇨implementa l'interfaccia java.util.ListIterator
  - ⇨gli oggetti vengono costruiti e forniti all'esterno dal metodo fabbricatore Iterator iterator() di LinkedList

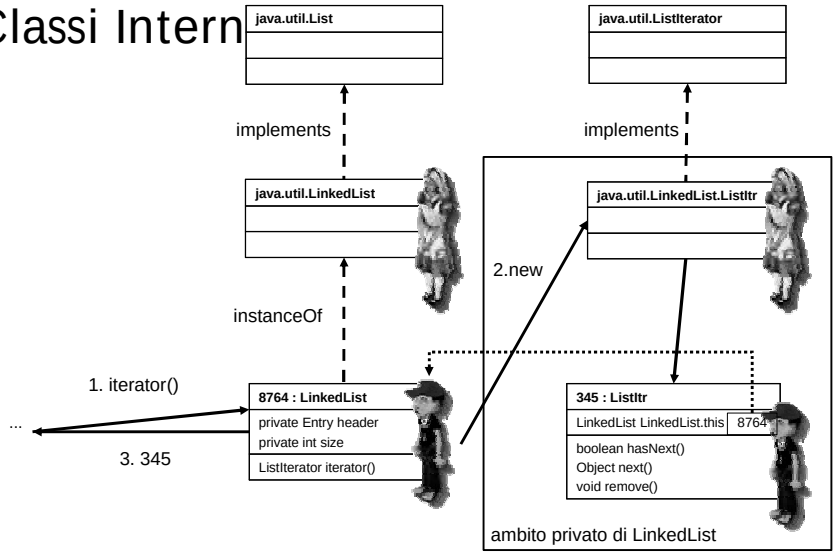
Classi Interne

- La particolarità della classe interna
  - ⇨per ciascun oggetto della classe interna c'è sempre un oggetto della classe contenitore
- Oggetto esterno
  - ⇨oggetto della classe esterna che ha creato l'oggetto della classe interna (oggetto interno)
  - ⇨l'oggetto interno mantiene un riferimento all'oggetto esterno

Classi Interne

- In altri termini
  - ⇨gli oggetti della classe interna privata possono essere costruiti solo da oggetti della classe esterna
  - ⇨l'oggetto della classe interna mantiene un riferimento speciale all'oggetto esterno che lo ha creato, per poterne usare i membri
  - ⇨il riferimento si chiama *ClasseEsterna*.this; se non c'è ambiguità può essere omesso

Classi Intern




```
package java.util;

public class LinkedList extends AbstractSequentialList
                        implements List, Cloneable, java.io.Serializable {

  // classe interna ListItr
  private class ListItr implements ListIterator {
     private Entry lastReturned = header;
     private Entry next;
     private int nextIndex;

     public boolean hasNext() {
        return nextIndex != size;
     }
     ...
  }

  private transient Entry header = new Entry(null, null, null);
  private transient int size = 0;

  public LinkedList() {
     header.next = header.previous = header;
  }
  ...
```

codice della classe interna LinkedList.ListItr

l'oggetto interno accede ai membri dell'oggetto esterno (equivalenti a LinkedList.this.header e LinkedList.this.size)

Classi Interne

- Attenzione
  - ⇨il legame tra oggetto interno e oggetto esterno non è una semplice associazione
  - ⇨a differenza dell'associazione ordinaria, attraverso il riferimento speciale l'oggetto interno può accedere anche ai membri privati dell'oggetto esterno (implementazione)
  - ⇨con l'associazione, viceversa, gli oggetti interagiscono solo attraverso l'interfaccia

Classi Interne

- Differenza con le classi nidificate
  - ⇨gli oggetti della classe nidificata non tengono riferimenti speciali ad oggetti della classe esterna
  - ⇨tipicamente sono gli oggetti della classe esterna a creare e ad accedere ai membri della classe nidificata (es: Entry)
  - ⇨nel caso delle classe interna tipicamente sono gli oggetti della classe interna che accedono a membri dell'oggetto esterno

Classi Interne

- In realtà
  - ⇨gli oggetti della classe interna non hanno senso se non nell'ambito di un oggetto esterno
- Per esempio
  - ⇨le classi interne non possono avere metodi statici (che sarebbero utilizzabili senza creato oggetti della classe, e quindi senza passare per l'oggetto esterno)

Implementazione

- In realtà
  - ⇨le classi nidificate e le classi interne sono un fenomeno rilevante solo per il compilatore
  - ⇨il compilatore le trasforma in classi ordinarie utilizzando piccoli accorgimenti
  - ⇨la macchina virtuale le tratta come classi ordinarie
- Gli accorgimenti usati dal compilatore
  - ⇨il nome e la gestione dell'accesso

Implementazione

- In nome delle classi membro
  - per ciascuna classe membro il compilatore crea un file .class che ha il nome NomeClasseEsterna$NomeClasseInterna.class
- Esempio: in rt.jar
  - LinkedList.class
  - LinkedList$Entry.class
  - LinkedList$ListItr.class

Implementazione

- Accorgimenti per l'accesso
  - il compilatore modifica leggermente il bytecode delle classi membro e della classe contenitore
  - aggiungendo metodi speciali che consentono la visibilità dei metodi privati da una parte e dall'altra
  - e aggiungendo un parametro al costruttore delle classi interne

Implementazione

- Una verifica
  - ⇨eseguiamo it.unibas.utilita.Ispezionatore sui tre file .class
- Tre aspetti significativi
  - ⇨la proprietà this$0 di LinkedList$ListItr
  - ⇨il costruttore di LinkedList$ListItr
  - ⇨i metodi statici access$xxx di LinkedList

```
>> java –cp ... it.unibas.utilita.Ispezionatore java.util.LinkedList
>> java –cp ... it.unibas.utilita.Ispezionatore java.util.LinkedList$ListItr
```


Implementazione

- Nota
  - ⇨i membri il cui nome contiene il simbolo $ sono considerati riservati dalla macchina virtuale
  - ⇨di conseguenza non possono essere chiamati in modo ordinario nel codice sorgente di altre classi
  - ⇨di conseguenza non rappresentano una violazione significativa del principio di incapsulamento

Classi Interne Anonime

- Un'ultima annotazione
  - ⇨il linguaggio Java consente anche di definire classi interne anonime
- Classe interna anonima
  - ⇨classe interna priva di nome
  - ⇨definita "in linea" nel codice di un metodo
  - ⇨in corrispondenza di un'istruzione di creazione di un oggetto

Classi Interne Anonime

- Alcuni vincoli
  - ⇨una classe interna anonima è una classe che implementa un'interfaccia o estende una superclasse
  - ⇨supponiamo che sia necessario creare oggetti della classe interna in una sola istruzione del codice
  - ⇨e che contestualmente sia necessario fare l'upcast al supertipo (interfaccia o superclasse)

Classi Interne Anonime

- Esempio
  - ⇨voglio creare e restituire un iteratore per la LinkedList nel metodo iterator()
  - ⇨e restituirlo come oggetto di tipo Iterator
- In questo caso
  - ⇨posso evitare di definire la classe interna ListItr e utilizzare una classe interna anonima come segue


```
package java.util;

public class LinkedList extends AbstractSequentialList
                        implements List, Cloneable, java.io.Serializable {

   private transient Entry header = new Entry(null, null, null);
   private transient int size = 0;

   public Iterator iterator() {
      return new Iterator() {
            private Entry lastReturned = header;
            private Entry next;
            private int nextIndex;
            public boolean hasNext() {
                return nextIndex != size;
            }
            public Object next() {...}
            public void remove() {...}
      }
   }
   ...
}
```

virtuale chiamata ad un costruttore del supertipo
NOTA: il costruttore in questione non esiste perchè java.util.Iterator è un'interfaccia

definizione della classe interna anonima

Classi Interne Anonime

- Nota
  - ⇨la definizione della classe, in questo caso, è addirittura locale ad un metodo
- La sintassi tipica della classe anonima

```
new Supertipo() {
    // definizione della class anonima
    <proprietà>
    <metodi>
}
```


Classi Interne Anonime

- Il file .class generato
  - ⇨sarebbe chiamato LinkedList$1.class
- Attenzione
  - ⇨le classi anonime peggiorano decisamente la leggibilità del codice
  - ⇨per stessa ammissione del creatore di Java – James Gosling – dovrebbero essere usate con parsimonia
  - ⇨e mai per classi di più di 6 righe di codice

Classi Interne Anonime

- In realtà
  - ⇨non sono uno strumento indispensabile
  - ⇨è sempre possibile utilizzare una classe interna con un nome esplicito al posto di una classe interna anonima
- Linea guida
  - ⇨limitare al minimo l'utilizzo delle classi interne anonime

Classi Interne Anonime

- Ma...
  - ⇨le classi anonime hanno un utilizzo "tipico" (ovvero rappresentano un idioma di Java)
- Utilizzo tipico delle classi interne anonime
  - ⇨per specificare un'implementazione fatta di un unico metodo di lunghezza relativamente breve
- Di conseguenza
  - ⇨il caso dell'iteratore non è un esempio tipico

Classi Interne Anonime

ATTENZIONE
alla lettura delle classi interne anonime

- Per capire l'utilizzo tipico
  - ⇨è possibile dare una lettura alternativa delle classi interne anonime
  - ⇨come uno strumento per "fornire un metodo concreto" in modo da poter implementare un'interfaccia
  - ⇨ovvero uno strumento per "associare un metodo concreto (argomento) ad un metodo astratto (parametro)"

Classi Interne Anonime

- Al solito
  - ⇨stiamo discutendo di "passaggio dei parametri" in un'accezione molto più ampia rispetto alla solita
  - ⇨accezione tradizionale: strumento per l'associazione tra dati
  - ⇨accezione alternativa (già vista): strumento per l'associazione tra tipi
  - ⇨ora: strumento per l'associazione tra metodi

Classi Interne Anonime

>> calcolatriceanonima

- Per vedere meglio questo utilizzo
  - ⇨vediamo un altro esempio
- La calcolatrice con le classi anonime
  - ⇨idea: costruisco un'interfaccia Calcolatrice, che prevede un metodo double operazione(double a, double b);
  - ⇨di volta in volta costruisco oggetti che sanno fare operazioni diverse associando al metodo operazione implementazioni diverse (somma, sottrazione, moltiplicazione)

Classi Interne Anonime

- L'interfaccia Calcolatrice
  - ⇨definisce la struttura di un oggetto capace di fare generiche operazioni aritmetiche su numeri reali
  - ⇨richiede di attribuire un'implementazione al metodo operazione()

```
package calcolatriceanonima;

public interface Calcolatrice {
   double operazione(double a, double b);
}
```

```
package calcolatriceanonima;

public class Principale {
   public static void main(String[] args) {
      Principale p = new Principale();
      p.eseguiOperazioni();
   }

   private int schermoMenu() {
      int scelta;
      System.out.println("--------------------------------");
      System.out.println("        Calcolatrice            ");
      System.out.println("--------------------------------");
      System.out.println("   1. Esegui somma");
      System.out.println("   2. Esegui differenza");
      System.out.println("   3. Esegui moltiplicazione");
      System.out.println("   0. Esci");
      System.out.print("   Scelta ----> ");
      scelta = it.unibas.utilita.Console.leggiIntero();
      while ((scelta < 0) || (scelta > 3)) {
         System.out.print ("   Errore. Ripeti ----> ");
         scelta = it.unibas.utilita.Console.leggiIntero();
      }
      return scelta;
   }
   ...
```



```
public void eseguiOperazioni() {
      Calcolatrice calcolatrice = null; boolean continua = true;
      while (continua) {
         double a = 0, b = 0;  int scelta = schermoMenu();
         if (scelta == 0) { continua = false; }
         else {
            System.out.print("   Primo argomento:   --> ");     a = it.unibas.utilita.Console.leggiDouble();
            System.out.print("   Secondo argomento: --> ");   b = it.unibas.utilita.Console.leggiDouble();
            if (scelta == 1) {
               calcolatrice = new Calcolatrice() {
                  public double operazione(double a, double b) {
                     return a + b;
                  }
               };
            } else if (scelta == 2) {
               calcolatrice = new Calcolatrice() {
                  public double operazione(double a, double b) {
                     return a - b;
                  }
               };
            } else if (scelta == 3) {
               calcolatrice = new Calcolatrice() {
                  public double operazione(double a, double b) {
                     return a * b;
                  }
               };
            }
            System.out.println("\n   Risultato: " + calcolatrice.operazione(a, b));
         }
      }
}
```

uso la classe interna anonima per associare ad operazione() un metodo che effettua somme

uso la classe interna anonima per associare ad operazione() un metodo che effettua differ.

uso la classe interna anonima per associare ad operazione() un metodo che effettua prodotti

Riassumendo

- Introduzione
- Classi Nidificate
- Classi Interne
- Implementazione
- Classi Interne Anonime

Termini della Licenza