Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa 08

Ereditarietà

A. Miola

Febbraio 2008

Contenuti

- Livelli di astrazione ed ereditarietà
- Estensione di classi
- Polimorfismo
- La classe Object
- Classi astratte
- Interfacce

Alcune riflessioni

- **Riprendiamo alcune riflessioni fatte in precedenza**
  - Cosa è il tipo `Object` ?
  - Che relazione c'è tra il tipo astratto *Q* dei numeri razionali e il tipo astratto Coppia ?
    - Il tipo *Q* è un caso particolare di Coppia
  - Che relazione c'è tra la classe `Razionale` e la classe `Coppia` ?
    - In che senso la classe `Razionale` può essere intesa come un caso particolare della classe `Coppia` ?
    - Cosa vuol dire in Java che una classe è un caso particolare di un'altra classe ?
  - Per rispondere a queste domande dobbiamo introdurre il concetto di ereditarietà e i relativi meccanismi di realizzazione

Ereditarietà

- Una delle caratteristiche distintive del paradigma di programmazione orientato agli oggetti è l'ereditarietà
  - l'ereditarietà permette di definire nuove classi mediante l'aggiunta e/o la specializzazione di funzionalità ad altre classi, già esistenti oppure appositamente progettate e definite
- Il linguaggio Java fornisce tre meccanismi (costrutti linguistici) di ereditarietà
  - estensione di classi
  - classi astratte
  - interfacce

Estensione di classi

- L'estensione (o derivazione) di classi è il meccanismo che permette di definire una classe (per istanziare oggetti) a partire da un'altra classe mediante l'aggiunta e/o la specializzazione di funzionalità
  - la classe "di partenza" è chiamata classe base oppure super-classe
  - la classe che viene definita è chiamata classe estesa o derivata oppure sotto-classe
  - tutte le proprietà e le operazioni definite dalla classe base sono implicitamente definite anche nella classe estesa che le eredita
    - la classe estesa può definire delle nuove proprietà e funzionalità per i propri oggetti
    - la classe estesa può ri-definire le funzionalità già definite nella classe base
  - ogni istanza della classe estesa può essere considerata anche una istanza della classe base

Esempio — la classe Persona

- Si consideri la seguente classe Persona
  - un oggetto Persona rappresenta una persona
  - l'unica proprietà di una Persona è il suo nome
    - nome è una variabile d'istanza (privata)
  - è possibile costruire una nuova Persona che ha un certo nome
    - la classe Persona ha un costruttore parametrico rispetto al nome
  - a una Persona è possibile chiedere il suo nome
    - il metodo (d'istanza) pubblico String getNome() restituisce il nome della Persona
  - a una Persona è possibile chiedere una sua descrizione
    - il metodo (d'istanza) pubblico String toString() restituisce un descrizione della Persona

La classe Persona . . .

```
/** Un oggetto Persona rappresenta una
  persona. */
class Persona {
    /** Il nome. */
    private String nome;

    /** Crea una Persona, dato il nome. */
    public Persona(String nome) {
        this.nome = nome;
    }
  . . . Segue . . .
```

. . . La classe Persona

```
. . . Segue . . .
/** Restituisce il nome della Persona. */
    public String getNome() {
        return nome;
    }

/** Restituisce una descrizione. */
    public String toString() {
        return "Mi chiamo " + getNome() + ".";
    }
}
```

Esempio — la classe Studente

- **Si supponga ora di voler definire la seguente classe Studente**
  - un oggetto Studente rappresenta uno studente universitario
    - le proprietà di uno Studente sono il suo nome e il nome dell'università in cui studia
  - è possibile costruire un nuovo Studente che ha un certo nome e studia in una certa università
    - la classe Studente ha un costruttore parametrico rispetto al nome e all'università
  - a uno Studente è possibile chiedere il suo nome e l'università in cui studia nonché una sua descrizione con i metodi
    - String getNome() che restituisce il nome dello Studente
    - String getUniversità() che restituisce l'università dello Studente
    - String toString() che restituisce un descrizione dello Studente

Dalla classe Persona alla classe Studente

- Si osservi come la classe Studente estenda e specializzi il comportamento della classe Persona
  - la classe Studente serve a modellare degli oggetti che sono "casi particolari" della classe Persona
    - un oggetto Studente sa eseguire tutte le operazioni degli oggetti Persona
    - un oggetto Studente sa eseguire anche delle ulteriori operazioni, che gli oggetti Persona non sanno eseguire
- In Java è possibile definire la classe `Studente` come estensione della classe `Persona`, di cui è una specializzazione

La classe Studente . . .

```
/** Un oggetto Studente rappresenta uno studente. */
class Studente extends Persona {
    /** L'università. */
    private String università;

    /** Crea uno Studente, dati nome e università. */
    public Studente(String nome, String università) {
        super(nome); // invoca il costruttore di Persona
        this.università = università;
    }

    /** Restituisce l'università dello Studente. */
    public String getUniversità() {
        return università;
    }
}
```

. . . La classe Studente

La classe estesa può ridefinire i metodi della classe base se ne vuole modificare la definizione (overriding o sovrascrittura) — ad esempio, toString()

```
/** Restituisce una descrizione dello Studente. */
    public String toString() {
        return "Mi chiamo " + getNome() + ". " +
                "Studio a " + getUniversità() + ".";
    }
```

Uso delle classi Persona e Studente . . .

- Gli oggetti della classe derivata sono compatibili con gli oggetti della classe base – ma non viceversa
- Il seguente esempio mostra come usare degli oggetti Persona e Studente

```
Persona mario = new Persona("Mario");
Studente paola = new Studente("Paola", "Roma Tre");
Persona p;
Studente s;
p = paola // OK! Studente e' compatibile con Persona
s = mario // NO! Persona non e' compatibile con Studente
```

. . . Uso delle classi Persona e Studente

```
System.out.println(mario.getNome()); // Mario
System.out.println(mario.toString());// Mi chiamo Mario.
/* OK! getNome e toString sono metodi di Persona */

System.out.println(paola.getNome());        // Paola
System.out.println(paola.getUniversità()); // Roma Tre
/* OK! getNome e getUniversità sono metodi di Studente */

System.out.println(mario.getUniversità());
/* NO! getUniversità e' un metodo di Studente */

System.out.println(paola.toString());
                    // Mi chiamo Paola. Studio a Roma Tre.
/* OK! toString e' un metodo di Studente
 * sovrascritto a quello omonimo di Persona */
```

Polimorfismo . . .

- **L'aspetto fondamentale dell'estensione di classe (e dell'ereditarietà in generale) è il polimorfismo**
  - per **polimorfismo** si intende la possibilità di assegnare il riferimento a un oggetto della classe estesa a una variabile il cui tipo è la classe base
    - vedi esempio nella precedente diapositiva 13
    - a tale oggetto è possibile richiedere solo il comportamento dichiarato dalla classe base
    - tuttavia, il comportamento dell'oggetto viene scelto sulla base del tipo effettivo dell'oggetto, e non sul tipo della variabile (**late binding**)

. . . Polimorfismo

- Ad esempio con rifetimento alle classi Persona e Studente

```
Persona paola;
  paola = new Studente("Paola", "Roma Tre");
  System.out.println(paola.toString());
        // Mi chiamo Paola. Studio a Roma Tre.
        // e non: Mi chiamo Paola.
        // toString e' definito anche in Persona
  System.out.println(paola.getUniversità());
        // NO, ERRORE (IN COMPILAZIONE)!
        // getUniversità non e' definito in Persona
```

Polimorfismo e parametri . . .

- In modo simile, nell'invocazione di un metodo che ha un parametro del tipo della classe base, è possibile usare come parametro attuale il riferimento a un oggetto istanza della classe estesa
  - a tale oggetto è possibile richiedere solo il comportamento dichiarato dalla classe base
  - tuttavia, il comportamento dell'oggetto viene scelto sulla base del tipo effettivo dell'oggetto (parametro attuale), e non sul tipo della variabile (parametro formale)

. . . Polimorfismo e parametri

Ad esempio

- definizione del metodo

```
public static void stampa(Persona p) {
      System.out.println(p.toString());
  }
```

- invocazione del metodo

```
ABC.stampa( new Studente("Marco", "Roma Tre") );
          // Mi chiamo Marco. Studio a Roma Tre.
```

Conversione esplicita . . .

- Si consideri nuovamente il caso di una variabile del tipo della classe base che memorizza il riferimento a un oggetto
  - se si è sicuri che la variabile memorizza il riferimento a un oggetto istanza della classe estesa, allora è possibile effettuare una conversione esplicita (cast) del riferimento, per ottenere un riferimento del tipo della classe estesa
    - la conversione permette di utilizzare il comportamento specifico della classe estesa
    - la conversione esplicita genera una eccezione se l'oggetto referenziato non ha il tipo della classe a cui si converte

. . . Conversione esplicita

- Ad esempio

```
Persona johnP;          // john come Persona
Studente johnS;         // john come Studente
johnP = new Studente("John", "Stanford");
johnS = (Studente) johnP;   // ok
System.out.println(johnS.getUniversità());// ok
System.out.println(johnP.getUniversità());// NO
```

La classe Object . . .

- In Java, tutte le classi estendono (direttamente o indirettamente) la classe Object definita nella API di Java (in java.lang)
  - la classe Object definisce un comportamento comune per tutti gli oggetti istanza
  - La classe Object è la super-classe per tutte le altre classi (già definite in Java o definite dall'utente)
  - tutte le classi ereditano questo comportamento comune — ma possono ridefinirlo se necessario

. . . La classe Object

- **Ad esempio, la classe Object definisce il metodo boolean equals(Object altro) per verificare se due oggetti sono uguali**
  - **nell'implementazione di Object, due oggetti sono uguali se sono identici (ovvero, se sono lo stesso oggetto)**
  - **ogni classe in cui è significativa una nozione di uguaglianza (diversa dall'identità) deve ridefinire questo metodo**
    - **si pensi ad esempio alla classe String**

Estensione di classi e progettazione di classi

- L'estensione di classi è una tecnica da utilizzare quindi nella progettazione di un insieme di classi che condividono alcune funzionalità
  - solitamente, si procede prima progettando la gerarchia delle classi che si vogliono definire, ciascuna con le proprie funzionalità, e poi implementando le varie classi, dalle super-classi verso le sotto-classi
  - per decidere se una classe può essere definita come l'estensione di un'altra classe, viene seguito il seguente criterio
    - la classe estesa deve modellare un insieme di oggetti che è un sottoinsieme degli oggetti modellati dalla classe base
    - si pensi ad esempio alle classi Persona e Studente

Il modificatore protected . . .

- Nella definizione di gerarchie di classi, l'uso dei soli modificatori public e private è troppo restrittivo
  - spesso è utile permettere a una classe estesa di accedere alle componenti "private" della classe base, senza che queste componenti siano rese accessibili a tutti gli altri oggetti
  - questa modalità di accesso non può essere definita utilizzando i modificatori public e private
- In questi casi può essere utilizzato il modificatore protected

. . . Il modificatore protected

- In questi casi può essere utilizzato il modificatore protected
  - un componente (variabile e metodo) dichiarato protected in una classe C può essere acceduto dalle classi che estendono C
  - Il modificatore protected offre un livello di restrizione dell'accesso intermedio tra public e private
  - nella definizione di classi da estendere, viene spesso utilizzato il modificatore protected anziché il modificatore private

Classi astratte . . .

- **Una classe astratta è una classe implementata in modo parziale**
  - una classe astratta è una classe che contiene la dichiarazione di alcuni metodi (metodi astratti), di cui viene specificata l'intestazione ma non il corpo
  - una classe astratta non può essere istanziata, appunto perché definita in modo incompleto
  - le classi astratte sono progettate per essere estese da classi che forniscono delle opportune implementazioni per i metodi astratti

. . . Classi astratte

- Le classi astratte sono utili nella definizione di una gerarchia di classi, in cui la super-classe (astratta) viene definita con i seguenti scopi
  - definire il comportamento comune per tutte le classi della gerarchia
  - dichiarare (senza implementare) le funzionalità che devono essere implementate da tutte le classi che la estendono

Esempio

- **Si vogliono definire delle classi i cui oggetti rappresentano delle forme geometriche**
  - **ad esempio, le classi Quadrato e Cerchio per rappresentare rispettivamente quadrati e cerchi**
  - **le caratteristiche delle forme geometriche sono le seguenti**
    - **i quadrati sono caratterizzati da un lato e da un colore**
    - **i cerchi sono caratterizzati da un raggio e da un colore**
    - **i quadrati devono saper calcolare il proprio lato, il proprio colore e la propria area**
    - **i cerchi devono saper calcolare il proprio raggio, il proprio colore e la propria area**
  - **Viene definita la classe (astratta) Forma che dichiara e/o definisce le caratteristiche comuni delle classi Quadrato e Cerchio**

La classe astratta Forma

```
/** Una forma geometrica. */
abstract class Forma {

    /** Il colore della forma. */
      protected String colore;
    /** Crea una nuova Forma di un certo colore. */
      public Forma(String colore) {
        this.colore = colore; }
    /** Restituisce il colore della forma. */
      public String colore() {
        return colore; }
    /** Restituisce l'area della forma. */
    public abstract double area();
    // ogni forma deve saper calcolare la propria area
}
```

La classe Quadrato

```
/** La forma geometrica quadrato. */
class Quadrato extends Forma {
    /** Il lato del quadrato. */
      protected double lato;
    /** Crea una nuovo Quadrato. */
      public Quadrato(double lato, String colore) {
        super(colore);
        this.lato = lato; }
    /** Restituisce il lato del quadrato. */
    public double lato() {
        return lato; }
    /** Restituisce l'area del quadrato. */
    // implementa il metodo astratto area()
    public double area() { return lato*lato; }
}
```

La classe Cerchio

```
/** La forma geometrica cerchio. */
class Cerchio extends Forma {
    /** Il raggio del cerchio. */
    protected double raggio;
    /** Crea una nuovo Cerchio. */
    public Cerchio(double raggio, String colore) {
        super(colore);
        this.raggio = raggio; }
    /** Restituisce il raggio del cerchio. */
    public double raggio() {
        return raggio; }
    /** Restituisce l'area del cerchio. */
    // implementa il metodo astratto area()
    public double area() { return raggio*raggio*Math.PI; }
}
```

Uso di forme geometriche

```
Quadrato q;   // un quadrato
Forma fq;     // un altro quadrato

q = new Quadrato(5, "bianco"); // lato 5 e colore bianco

fq = new Quadrato(10, "rosso"); // lato 10 e colore rosso

System.out.println(q.area());      // 25
System.out.println(q.colore());    // bianco
System.out.println(q.lato());      // 5

System.out.println(fq.area());     // 100
System.out.println(fq.colore());   // rosso
System.out.println(fq.lato());     // NO, errore
                                   // di compilazione!
```

Classi astratte e polimorfismo

❑ Caratteristiche dell'esempio mostrato

- la classe Forma è stata dichiarata abstract perché contiene la dichiarazione del metodo astratto area()
  - la classe astratta Forma non può essere istanziata direttamente
- le classi Quadrato e Cerchio sono "concrete" perché estendono la classe Forma e ne implementano tutti i metodi astratti
  - le classi Quadrato e Cerchio possono essere istanziate

❑ Una variabile di tipo Forma

- può memorizzare il riferimento a una forma (un quadrato o un cerchio)
- può essere usata per invocare un metodo definito da Forma — ad esempio, colore()
- può essere usata per invocare un metodo astratto dichiarato da Forma — ad esempio, area()

Interfacce

- In Java, una interfaccia (interface) è una unità di programmazione che consiste nella dichiarazione di un certo numero di metodi d'istanza e pubblici, che sono implicitamente astratti
  - in prima approssimazione, una interfaccia è simile a una classe astratta che dichiara solo metodi astratti, senza fornire alcuna implementazione
- Una classe implementa una interfaccia se implementa (definisce) tutti i metodi d'istanza dichiarati dall'interfaccia

Esercizi

- Riesaminare le domande poste inizialmente circa il tipo Razionale e il tipo Coppia
- Definire il tipo Coppia e il tipo Razionale utilizzando i meccanismi di ereditarietà visti
- Definire il tipo Complesso come sottotipo del tipo Coppia

Riferimenti

- Per ulteriori approfondimenti si può fare riferimento al Capitolo 25 del libro di testo