Programmazione in Java e gestione della grafica

Lezione 15

TEST

- Bene (in media) per il TestFeb !

- TestMar: 25 marzo 2013

Esercizio

- A partire dall'esercizio per casa del TestFeb definire una classe StudenteP che eredita Studente e che contiene il voto del progetto.
- Definire i costruttori nella superclasse e in studenteP.
- Il progetto avrà una valutazione da 0 a 3 punti da aggiungere al voto calcolato con la media. Se lo studente non fa il progetto gli vengono sottratti 2 punti dal voto finale.
- La classe dovrà avere un metodo votoFinale che aggiunge alla media il voto del progetto.
- Riscrivere il programma principale in cui si immettono i voti dei test e dei progetti di tutti (30) gli studenti del primo anno (valori random?) e si calcolano i voti finali degli studenti. Assumiamo di utilizzare la media che esclude il voto più basso. Si stampa sullo schermo la tabella con tutti i voti + la colonna contenente "promosso" o "bocciato". Si calcola inoltre il numero di studenti promossi nella sessione.

Ereditarietà (Inheritance)

- Riusabilità del Software
- Creare nuove classi da classi già esistenti
  - "Assorbo"le classi e i metodi delle classi esistenti
  - Le estendo con nuove "capacità"
- Una Subclasse **estende** la sua superclasse
  - Creo oggetti più specializzati
  - Eredito metodi dalla sulerclasse
    - » Li posso adattare meglio
  - Aggiungo metodi e campi addizionali

Gerarchia delle classi

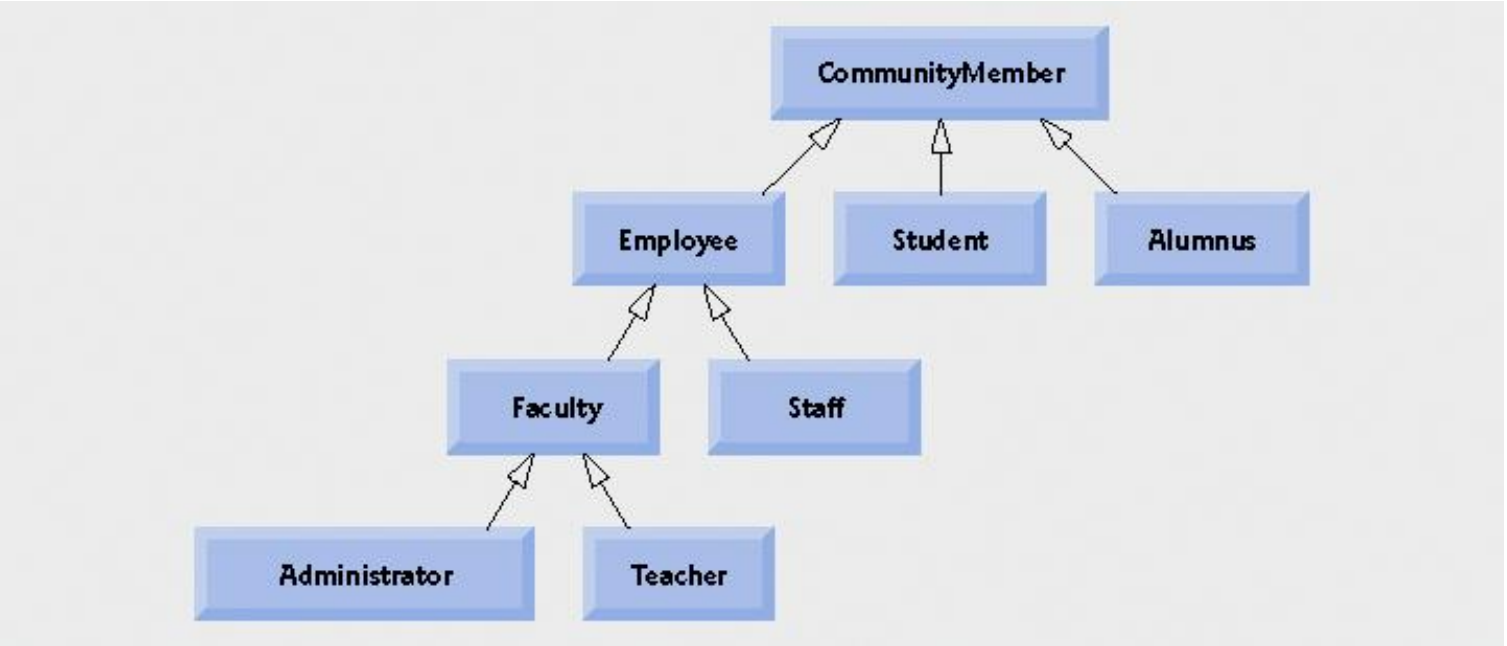
- Superclasse Diretta
  - Un livello più in alto
- Superclasse Indiretta
  - Due o più livelli più in alto
- Eredità singola
  - Ad 1 livello più su c'e' solo una superclasse

Superclassi e subclassi

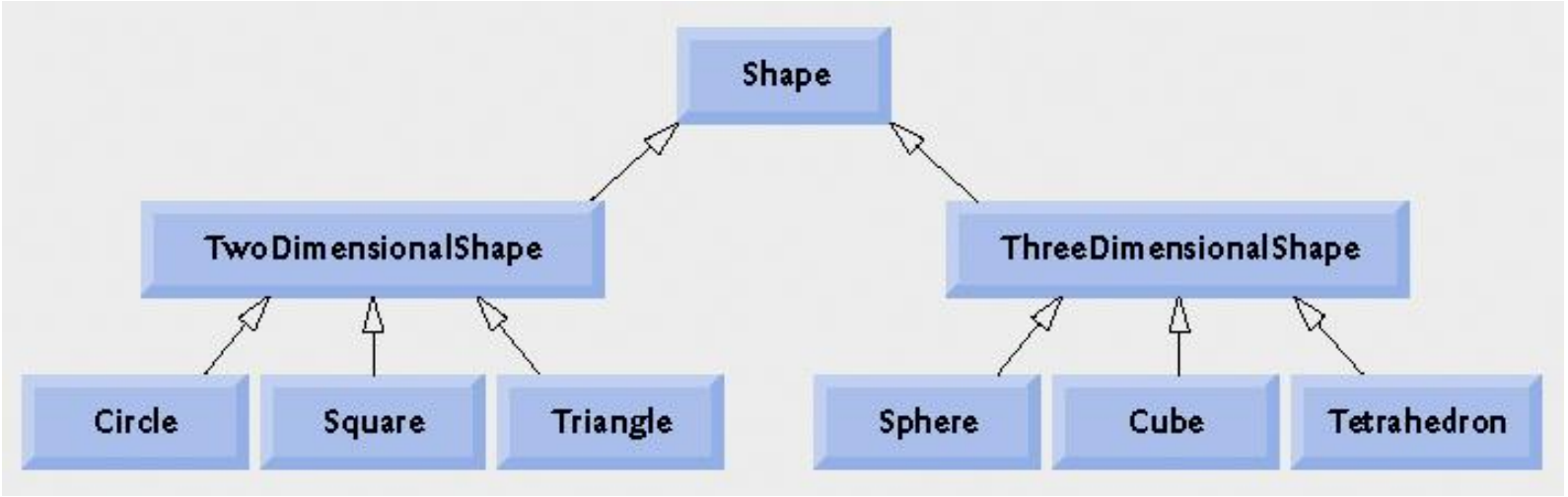
- Superclasses and subclasses
  - Object of one class “is an” object of another class
    - Example: Rectangle is quadrilateral.
      - Class `Rectangle` inherits from class `Quadrilateral`
      - `Quadrilateral`: superclass
      - `Rectangle`: subclass
  - Superclass typically represents larger set of objects than subclasses
    - Example:
      - superclass: `Vehicle`
        - » Cars, trucks, boats, bicycles, …
      - subclass: `Car`
        - » Smaller, more-specific subset of vehicles

Superclassi e subclassi

- Gerarchia di Ereditarietà ha una struttura ad albero.
  - Ogni classe diventa
    - superclasse
      - Fornice membri alle classi “sotto”
    - subclasse
      - Eredita membri dalle classi “sopra”



Gerarchia di ereditarietà per la classe comunitaUniversitaria



Gerarchia di ereditarietà per la classe shape

Membri con accesso `protected`

- Livello Intermedio di protezione tra `public` e `private`
- I membri `protected` sono accessibili da
  - Membri delle superclassi
  - Membri delle sottoclassi
  - Membri di classi dello stesso package
- per accedere da una sottoclasse ad un membro della superclasse
  - Keyword `super` e (.)

Osservazione

- Metodi di una sottoclasse non possono accedere direttamente a membri `private` della loro superclasse. Una sottoclasse può cambiare una variabile `private` della sua superclasse soltanto attraverso metodi non-`private` che stanno nella superclasse e sono ereditati dalla sottoclasse.

Osservazione

- E’ utile dichiarare le variabili come `private`. Questo aiuta sia in fase di test e debug che di modifica dei sistemi.

Esempio (dal libro)

- `CommissionEmployee/ BasePlusCommissionEmployee`
  - `CommissionEmployee`
    - First name, last name, SSN, commission rate, gross sale amount
  - `BasePlusCommissionEmployee`
    - First name, last name, SSN, commission rate, gross sale amount
    - Base salary

ESEMPIO (dal libro di testo) Classe `CommissionEmployee`

- Classe `CommissionEmployee`
  - Estende classe `Object`
    - Keyword `extends`
    - Ogni classe in Java estende una classe esistente
      - Eccetto `Object`
    - Ogni classe eredita I metodi di `Object`

Osservazione

- Il compilatore Java imposta la superclasse di una classe a `Object` tutte le volte che nella dichiarazione della classe non si estende esplicitamente una superclasse.

Classe `Object`

- Metodi della classe `Object`
  - `clone`
  - `equals`
  - `finalize`
  - `getClass`
  - `hashCode`
  - `notify`, `notifyAll`, `wait`
  - `toString`

Method	Description
`clone`	This `protected` method, which takes no arguments and returns an `Object` reference, makes a copy of the object on which it is called. When cloning is required for objects of a class, the class should override method `clone` as a `public` method and should implement interface `Cloneable` (package `java.lang`). The default implementation of this method performs a so-called shallow copy—instance variable values in one object are copied into another object of the same type. For reference types, only the references are copied. A typical overridden `clone` method's implementation would perform a deep copy that creates a new object for each reference type instance variable. There are many subtleties to overriding method `clone`. You can learn more about cloning in the following article: `java.sun.com/developer/JDCTechTips/2001/tt0306.html`

Fig. 9.18 | `Object` methods that are inherited directly or indirectly by all classes. (Part 1 of 4)

Method	Description
`Equals`	**This method compares two objects for equality and returns `true` if they are equal and `false` otherwise. The method takes any `Object` as an argument. When objects of a particular class must be compared for equality, the class should override method `equals` to compare the contents of the two objects. The method's implementation should meet the following requirements:** **• It should return `false` if the argument is `null`.** **• It should return `true` if an object is compared to itself, as in `object1.equals( object1 )`.** **• It should return `true` only if both `object1.equals( object2 )` and `object2.equals( object1 )` would return `true`.** **• For three objects, if `object1.equals( object2 )` returns `true` and `object2.equals( object3 )` returns `true`, then `object1.equals( object3 )` should also return `true`.** **• If `equals` is called multiple times with the two objects and the objects do not change, the method should consistently return `true` if the objects are equal and `false` otherwise.** **A class that overrides `equals` should also override `hashCode` to ensure that equal objects have identical hashcodes. The default `equals` implementation uses operator == to determine whether two references *refer to the same object* in memory. Section 29.3.3 demonstrates class `String`'s `equals` method and differentiates between comparing `String` objects with == and with `equals`.**

Fig. 9.18 | `Object` methods that are inherited directly or indirectly by all classes. (Part 2 of 4)

Method	Description
`finalize`	This `protected` method (introduced in Section 8.10 and Section 8.11) is called by the garbage collector to perform termination housekeeping on an object just before the garbage collector reclaims the object's memory. It is not guaranteed that the garbage collector will reclaim an object, so it cannot be guaranteed that the object's `finalize` method will execute. The method must specify an empty parameter list and must return `void`. The default implementation of this method serves as a placeholder that does nothing.
`getClass`	Every object in Java knows its own type at execution time. Method `getClass` (used in Section 10.5 and Section 21.3) returns an object of class `Class` (package `java.lang`) that contains information about the object's type, such as its class name (returned by `Class` method `getName`). You can learn more about class `Class` in the online API documentation at `java.sun.com/j2se/5.0/docs/api/java/lang/Class.html`.

Fig. 9.18 | `Object` methods that are inherited directly or indirectly by all classes. (Part 3 of 4)

Method	Description
`hashCode`	A hashtable is a data structure (discussed in Section 19.10) that relates one object, called the key, to another object, called the value. When initially inserting a value into a hashtable, the key's `hashCode` method is called. The hashcode value returned is used by the hashtable to determine the location at which to insert the corresponding value. The key's hashcode is also used by the hashtable to locate the key's corresponding value.
`notify`, `notifyAll`, `wait`	Methods `notify`, `notifyAll` and the three overloaded versions of `wait` are related to multithreading, which is discussed in Chapter 23. In J2SE 5.0, the multithreading model has changed substantially, but these features continue to be supported.
`toString`	This method (introduced in Section 9.4.1) returns a `String` representation of an object. The default implementation of this method returns the package name and class name of the object's class followed by a hexadecimal representation of the value returned by the object's `hashCode` method.

Fig. 9.18 | `Object` methods that are inherited directly or indirectly by all classes. (Part 4 of 4)

Esempio direttamente al compilatore

dal libro di testo cap. 9