Gerarchia di classi Java

Parte 2
Programmazione
Corso di laurea in Informatica

Costruttori della sottoclasse

- I costruttori non vengono ereditati, non essendo membri di classe
- Vengono definiti esplicitamente dal programmatore
- Oppure viene usato quello di default
  - Il costruttore di default è quello senza argomenti fornito automaticamente quando non ne avete definito uno

Chiamata al costruttore della sottoclasse

- Quando si istanzia un oggetto, si avvia la sequenza d'azioni:
  1. Viene allocato lo spazio per il nuovo oggetto sullo heap. Tutti i campi dell'istanza vengono inizializzati a valori di default.
  2. Se esiste si esegue la chiamata esplicita (`this()` con o senza parametri) a un costruttore della stessa classe
  3. Altrimenti si invoca il costruttore esplicito o implicito della superclasse e si inizializzano i campi d'istanza nell'ordine in cui appaiono nel codice.
  4. Si esegue il resto del codice del costruttore
  5. I passi da 2. a 4. vengono ripetuti ricorsivamente per tutte le classi della gerarchia.

Ordine della chiamata al costruttore

- Java richiede che sia completamente inizializzato l'oggetto che descrive il padre prima che si possa eseguire una qualunque computazione sul figlio
- Chiamata implicita al costruttore della superclasse
  - `Arte.java`, `Disegno.java` e `Schizzo.java`
- Chiamata esplicita al costruttore della superclasse tramite `super`
  - `Libro_2.java`, `Dizionario_2.java` con `TestDizionario_2.java`

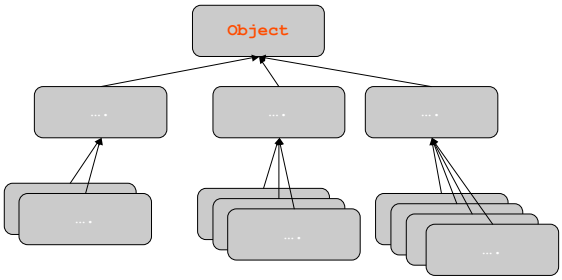
Costruzione dell'oggetto della sottoclasse

- Di default ogni costruttore di sottoclasse chiama il costruttore senza argomenti del padre
- Se il padre non ne ha uno la sottoclasse non viene compilata
- Se la superclasse ha costruttori con parametri, si deve invocare `super` con i parametri appropriati per controllare esplicitamente la costruzione dell'oggetto padre
- La chiamata a `super` deve essere la prima istruzione del costruttore

Ereditarietà singola e multipla

- Java fornisce solo ***ereditarietà singola***
  - La sottoclasse deriva da una sola superclasse
- ***L'ereditarietà multipla*** consentirebbe la derivazione di una classe da più classi antenati, e di ereditare i membri di più classi
- L'ereditarietà multipla introduce però ambiguità, che per essere risolte renderebbero Java più complicato
  - Ad esempio: lo stesso nome di variabile in più classi
- Nella maggior parte dei casi, l'uso di ***interfacce*** supera i limiti dell'ereditarietà singola senza le complicazioni di quella multipla

Diagramma delle classi



La classe Object

- La classe `Object` è predefinita nel pacchetto `java.lang` della libreria standard di classi
- Tutte le classi discendono dalla classe `Object`
- Se una classe non è esplicitamente dichiarata discendente da una superclasse, si assume che discenda dalla classe `Object`
- La classe `Object` è la radice di tutti gli alberi di discendenza

La classe Object

- La classe `Object` contiene alcuni metodi, pochi ma utili, che vengono ereditati da tutte le classi
- Il metodo `toString` ad esempio è definito per la classe `Object`
  - Il metodo `toString` della classe `Object` restituisce una stringa di caratteri contenente il nome della classe e un valore che identifica l'oggetto univocamente
- Ogni volta che definiamo esplicitamente `toString`, in effetti lo riscriviamo

La classe Object

- Il metodo `println` richiama il metodo `toString` per ogni oggetto che gli viene passato come parametro
  - perché tutti gli oggetti hanno un metodo `toString` ottenuto per ereditarietà
- Il metodo `equals` della classe Object determina se due riferimenti sono *alias*
  - Spesso è necessario riscrivere `equals` per definire il metodo con una semantica appropriata
- `Studente.java`, `StudenteDott.java` con `Accademia.java`

Decomposizione di metodi e ereditarietà

- Nella decomposizione di metodi spesso si usano metodi di servizio richiamati all'interno di un metodo, in genere dichiarati `private`
- Se il metodo di servizio deve essere ereditato e sovrascritto va posta attenzione ai metodi interni e alla loro accessibilità
- `Figure.java` superclasse `Triangle.java`, `Box.java` e `Diamond.java` `Driver.java`

Riferimenti ed ereditarietà

- Tramite riferimenti si può accedere a un oggetto in una relazione di ereditarietà tra classi.
- Se la classe `Festa` ha una sottoclasse `Natale`, allora una variabile a riferimento della classe `Festa` può anche essere usata per puntare a un oggetto `Natale`


```
Festa giorno;
giorno = new Natale();
```

Riferimenti e ereditarietà

- Nella gerarchia di classi la sottoclasse specializza la superclasse e introduce nuove funzionalità
- La nuova classe può essere vista come un tipo particolare di quella esistente
- Questo consente l'upcasting
  - La nuova classe è costituita dagli elementi della classe base cui aggiunge i propri
  - L'upcasting è sempre lecito


```
class Strumento {
      public void suona() {…}
      static void brano (Strumento s) {
            …
            s.suona()
      }
}
Class Archi extends Strumento {
      …
      public static void main(String[] a) {
            Archi violino = new Archi();
            Strumento.brano(violino);
      }
}
```

Al metodo di classe `brano` viene passato un parametro di tipo `Archi` e non di tipo `Strumento`.
Il riferimento `violino` a un oggetto di classe `Archi` viene convertito a un riferimento alla classe `Strumento`, mediante un ***upcasting***.

Riferimenti ed ereditarietà

- Possiamo assegnare a variabili a riferimento di una superclasse un oggetto di una sottoclasse, semplicemente con un assegnamento
  - Conversione ***larga*** o upcasting
- Assegnare a un oggetto discendente il riferimento a un antenato può essere fatto, ma richiede un ***cast*** esplicito
  - Conversione ***stretta*** o downcasting
- La conversione larga è più utile e meno pericolosa

Polimorfismo via ereditarietà

- Un *riferimento polimorfico* punta a oggetti di tipo diverso in tempi diversi
- Le interfacce possono essere usate per creare riferimenti polimorfici
- L'ereditarietà può essere usata come base del polimorfismo
  - Una variabile a riferimento può puntare a un oggetto in un dato istante e puntarne a un altro (legato al primo nella struttura gerarchica delle classi) in un altro momento

Polimorfismo via ereditarietà

- Sia `celebrare` un metodo della classe `Festa`, sovrascritto dalla classe `Natale`
- L'invocazione

  `giorno.celebrare();`

  produce effetti diversi:
  - Se `giorno` si riferisce a un oggetto della classe `Festa`, viene invocata la versione `celebrare()` della classe `Festa`; se invece punta a un oggetto della sottoclasse `Natale` verrà invocata la versione specifica e sovrascritta della sottoclasse

Polimorfismo via ereditarietà

- È il tipo dell'oggetto cui si punta, che determina quale metodo viene invocato
- Si noti che, se l'invocazione avviene in un ciclo, la stessa riga di codice potrebbe eseguire metodi diversi a ogni passo
- I riferimenti polimorfici vengono risolti a ***run-time***, e non durante la compilazione

Classi astratte

- Una classe incompleta che contiene metodi astratti
- È un segnaposto nella gerarchia delle classi, modella un concetto generico
  - Il modificatore `abstract` nell'intestazione della classe
- Una classe `abstract` è incompleta per definizione
- La classe `abstract` non può essere instanziata ma deve essere specializzata

Classi astratte

- Una classe deve essere dichiarata astratta se:
  - Contiene la dichiarazione di un metodo astratto
  - Eredita un metodo astratto dalla superclasse, non ne fornisce l'implementazione che demanda ad una sua sottoclasse
  - Dichiara di implementare un interfaccia, ma non fornisce l'implementazione di tutti i metodi

Classi astratte

- Il discendente di una classe astratta deve sovrascrivere i metodi astratti della superclasse da cui discende o sarà considerata anche lei astratta
- Un metodo astratto non può essere definito `final`, `static` o `private`
  - Non `final`, perché dovrà essere sovrascritto
  - Non `static`, perchè non ha ancora una definizione
  - Non `private`, perchè non potrebbe essere ereditato
- Una questione di scelta di design
  - Aiuta a stabilire gli elementi comuni che dovranno avere le sottoclassi di una classe che rappresenta un concetto astratto

Quando sono appropriate

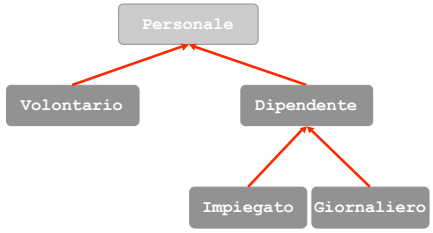
- Iniziare una gerarchia di classi che devono essere specializzate ulteriormente
  - Qui si usa la classe astratta per contenere metodi la cui specializzazione è rinviata alle classi specializzate
  - Definire il comportamento fondamentale della classe senza darne l'implementazione
- Non si devono istanziare oggetti della classe. La dichiarazione `abstract` garantisce che non verrà mai istanziata.
- I dati comuni possono essere raccolti in una superclasse, anche senza metodi.
  - Qui si usa una gerarchia di classi in cui una classe serve per scambiare dati ma non ci sono metodi comuni

Classe astratte – sintesi

- Una classe astratta è definita con il modificatore `abstract`
- Può contenere metodi implementati e costruttori.
- Se ha almeno un metodo astratto allora deve essere dichiarata astratta.
- Non è possibile istanziarla. Spesso si dichiara una variabile di tipo classe astratta, in cui successivamente si archiviano riferimenti a oggetti delle sottoclassi
- Le sottoclassi devono fornire implementazione per tutti i metodi astratti o anche loro saranno considerate astratte.
- I metodi astratti non possono essere dichiarati con i modificatori `private`, `static` o `final`.

Classi astratte e polimorfismo via ereditarietà

- Data la seguente struttura gerarchica:



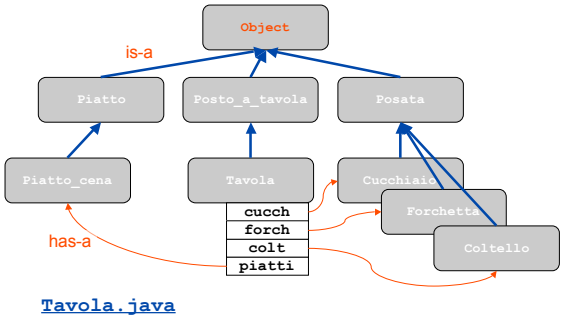
Un esempio

- Si risolve il compito di calcolare la retribuzione …
- `Istituzione.java` driver con il main
- `Staff.java` inizializza la lista del personale
- `Personale.java` una classe astratta
- `Dipendente.java` modella un dipendente generico
- `Volontario.java` modella un volontario
- `Impiegato.java` modella un impiegato che può prendere gratifiche, specializzando un dipendente
- `Giornaliero.java` modella un lavoratore a giornata, specializzando un dipendente generico

Accesso indiretto

- Tutti i membri della superclasse sono accessibili alla sottoclasse, anche se non sono ereditati
- I membri di classe ereditati sono accessibili direttamente
- I membri non ereditati sono accessibili indirettamente tramite i metodi ereditati della superclasse
- `Calorie.java` driver, istanzia oggetti di classe `Pizza` e `Torta`
- `Cibo.java` istanzia un oggetto con dati grammi di grasso e calcola le calorie dovute al grasso
- **`Pizza.java`** e **`Torta.java`**

Composizione di classi vs ereditarietà



`Tavola.java`

Gerarchie di interfacce

- Il meccanismo dell'ereditarietà può essere applicato alle interfacce
  - Un'interfaccia può diventare l'antenata di un'altra
  - L'interfaccia figlio eredita tutti i metodi astratti dal genitore
- Una classe che implementa un'interfaccia derivata deve definire tutti i metodi dell'interfaccia genitore oltre a quelli dell'interfaccia figlio
- Le gerarchie di classi e quelle di interfacce sono distinte

Esercizio

- Creare le classi **A** e **B** con costruttori di default (senza parametri) che segnalano son un messaggio l'avvenuta creazione
- Definire una terza classe **C** derivata da **A**, e create un membro **B** all'interno del codice di **C**.
- Non definite un costruttore per **C**.
- Create un oggetto di C e osservate.

Esercizio modificato

- Modificate l'esercizio precedente introducendo dei costruttori con argomento per **A** e **B**
- Definite un costruttore per **C** e eseguite le inizializzazioni necessarie all'interno del costruttore di **C**