UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Metodologie di Programmazione

Lezione 14: Ereditarietà (parte 2)

Lezione 14: Sommario

UNITELMA SAPIENZA

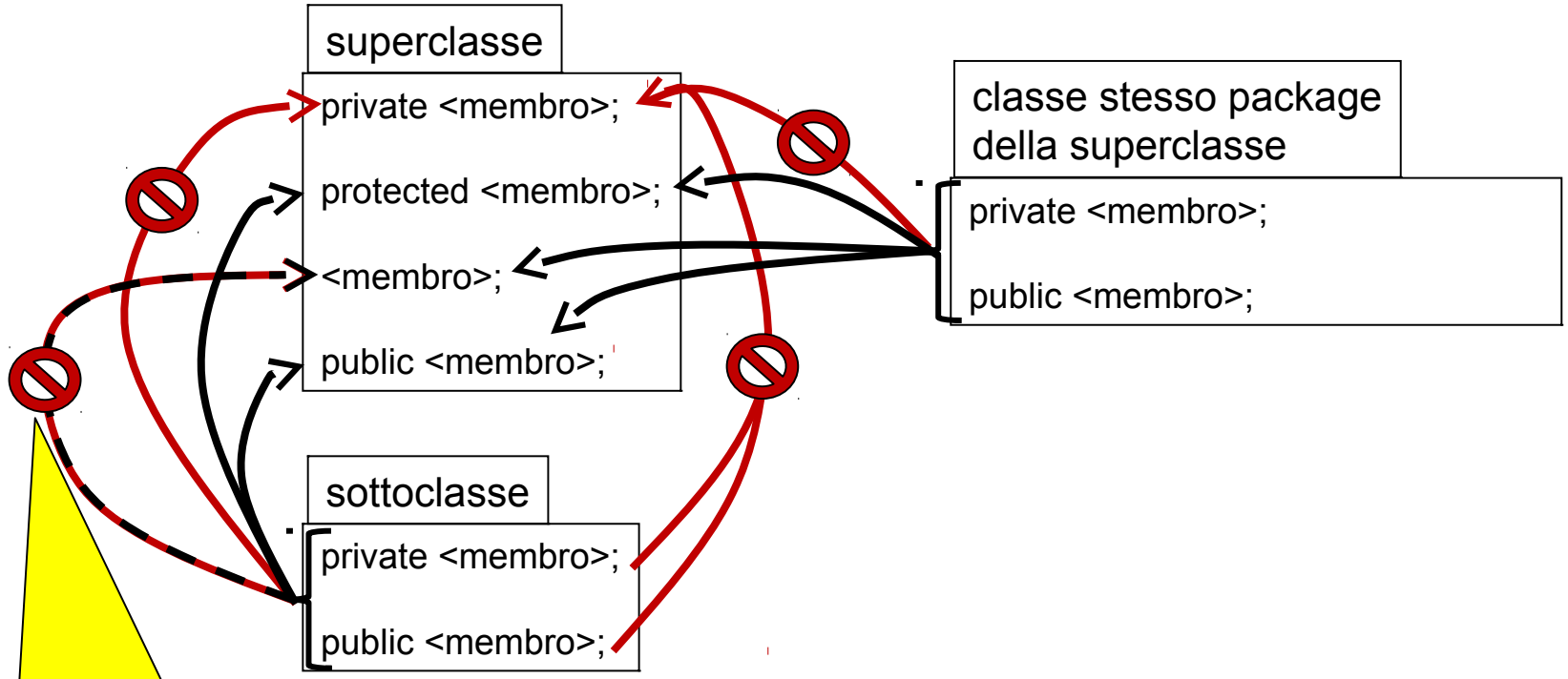
SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Modificatori di visibilità
- Ereditarietà vs. composizione
- Esempi ed esercizi

In dettaglio sulla visibilità (1)

- Quattro possibilità per campi e metodi:
  - Private: visibile solo all'interno della classe
  - Public: visibile a tutti
  - Default: visibile all'interno di tutte le classi del package
  - Protected: visibile all'interno di tutte le classi del package e delle sottoclassi (indipendentemente dal package)

In dettaglio sulla visibilità (2)



Accessibile se la sottoclasse è nello stesso package

Is-a contro has-a

- E' **molto** importante distinguere tra relazioni di tipo è-un (is-a) e relazioni di tipo ha-un (has-a)
- Is-a rappresenta l'ereditarietà
  - Un oggetto di una sottoclasse può essere trattato come un oggetto della superclasse
  - **Domanda:** la sottoclasse è-un superclasse? (es. Paperino è un PersonaggioDisney? Sì! QuiQuoQua è un Paperino? No!)
- Has-a rappresenta la composizione
  - Un oggetto contiene come membri **riferimenti** ad altri oggetti
  - **Domanda:** un oggetto contiene altri oggetti? (es. Bagno contiene Vasca? Sì! PersonaggioDisney contiene Paperino? No!)

BarraDiEnergia & BarraConPercentuale

- Creare una classe BarraDiEnergia costruita con un intero k che ne determina la lunghezza massima. Inizialmente la barra è vuota. La barra è dotata di un metodo per l'incremento unitario del suo livello di riempimento e di un metodo toString che ne fornisca la rappresentazione sotto forma di stringa (es. se il livello è 3 su 10, la stringa sarà "OOO========".
- Creare quindi una seconda classe BarraDiEnergiaConPercentuale che fornisce una rappresentazione sotto forma di stringa come BarraDiEnergia ma stampando in coda alla stringa la percentuale del livello di riempimento. Per esempio, se il livello è 3 su 10, la stringa sarà "OOO======== 30%".

Esercizio: ListaDiInteri e ListaOrdinataDiInteri

- Implementare una classe ListaDiInteri mediante un array (con i metodi specificati in fondo alle diapositive della parte: "controllo e array")
- Implementare quindi una classe ListaOrdinataDiInteri per creare liste di interi ordinati in modo crescente. Oltre agli altri metodi di ListaDiInteri (esclusi quelli di aggiunta) essa definisce i seguenti 3 metodi di aggiunta:
  - Aggiungi un intero in coda alla lista: l'aggiunta avviene solo se l'intero preserva l'ordine della lista.
  - Aggiungi un intero nella posizione specificata: come sopra, l'aggiunta avviene solo se l'intero preserva l'ordine degli interi della lista
  - Aggiungi un intero: l'intero viene inserito nella posizione appropriata, in modo da preservare l'ordine degli interi della lista
- L'array non deve essere ordinato con metodi di sorting, quali Arrays.sort (né vostri metodi di sorting *completo* dell'array)
- **Extra:** permettere di specificare un parametro da passare opzionalmente al costruttore di ListaOrdinataDiInteri per stabilire l'ordine della lista (crescente o decrescente; per default, crescente)

Esercizio: Animali

- Progettare la classe Animale che rappresenti un generico animale
- La classe possiede i metodi emettiVerso() e getNumeroDiZampe()
- Possiede inoltre il metodo getTaglia() che restituisce un valore scelto tra: piccola, media e grande.
- Progettare quindi le classi Mammifero, Felino, Gatto (taglia piccola), Tigre (grande), Cane, Chihuahua (piccola), Beagle (media), Terranova (grande), Uccello, Corvo (media), Passero (piccola), Millepiedi (piccola)
- Personalizzare in modo appropriato la taglia, il numero di zampe e il verso degli animali

Esercizio: Conto bancario

- Progettare la classe ContoBancario che rappresenti un conto con informazioni relative al denaro attualmente disponibile, il codice IBAN
- Modellare quindi una generica operazione bancaria Operazione che disponga di un metodo esegui()
- Modellare quindi i seguenti tipi di operazione:
  - PrelevaDenaro: preleva una specificata quantità di denaro da un dato conto
  - SvuotaConto: preleva tutto il denaro da un dato conto
  - VersaDenaro: versa del denaro sul conto specificato
  - SituazioneConto: stampa l'attuale saldo del conto
  - Bonifico: preleva del denaro da un conto e lo versa su un altro

Esercizio: Distributore di bevande

- Progettare una classe Prodotto con un prezzo e tre tipi diversi di prodotto: Caffè, Cappuccino, Cioccolato
- Progettare la classe DistributoreDiBevande che rappresenti un distributore automatico costruito con un intero N che determina il numero di prodotti nel distributore
- La classe prevede i seguenti metodi:
  - un metodo carica() che inserisce N prodotti di tipo e ordine casuale
  - un metodo inserisciImporto() che permette di inserire un importo nella macchinetta
  - un metodo getProdotto() che, dato in ingresso un numero di prodotto, restituisca il prodotto associato a quel numero e decrementi il saldo disponibile nel distributore
  - un metodo getSaldo() che restituisca il saldo attuale del distributore
  - un metodo getResto() che restituisca il resto dovuto e azzeri il saldo

Esercizio: Espressioni matematiche

- Progettare una serie di classi che modellino le espressioni matematiche secondo la seguente definizione:
  - Una costante di tipo double è un'espressione
  - Una variabile con nome di tipo stringa e valore double è un'espressione
  - Se e_1 è un'espressione, allora $-e_1$ è un'espressione
  - Se e_1, e_2 sono espressioni, allora e_1 op e_2 è un'espressione dove op può essere l'operatore +, -, *, /, %
- Ogni tipo di espressione (costante, variabile, espressioni composte) deve essere modellata mediante una classe separata
- Ogni espressione dispone del metodo getValore() che restituisce il valore che quell'espressione possiede in quel momento
- Costruire quindi l'espressione –(5+(3/2)-2)*x e calcolarne il valore quando la variabile x vale 3 e quando la variabile x vale 6

Esercizio: il Gioco dell'Oca

- Progettare il Gioco dell'Oca modellando:
  - Il Giocatore, che mantiene l'informazione sulla posizione nel tabellone e i punti accumulati e implementa il metodo tiraDadi()
  - Il Tabellone come sequenza di caselle costruita a partire da un intero N e da un elenco di giocatori; la classe dispone dell'operazione di posizionamento dei giocatori tenendo conto dell'effetto "gioco dell'oca" in cui, arrivati alla fine, si torna indietro
- Diversi tipi di caselle ciascuna con un diverso effetto a seguito del posizionamento del giocatore su quella casella:
- una CasellaVuota (nessun effetto sul giocatore)
- una CasellaSpostaGiocatore che sposta il giocatore di x caselle (avanti se x > 0 o indietro se x < 0)
- una CasellaPunti che ha l'effetto di far guadagnare o perdere un certo numero di punti al giocatore
- la classe GiocoDellOca che, dato un intero N e dati i giocatori, che inizializza un tabellone di lunghezza N e implementa il metodo giocaUnTurno() che fa effettuare una mossa a ognuno dei giocatori

Esercizio: Tetris

- Progettare il gioco del Tetris modellando la classe Pezzo con le seguenti operazioni:
  - Left : sposta a sinistra il pezzo
  - Right: sposta a destra il pezzo
  - Rotate: ruota il pezzo in senso orario
  - Down: manda giù il pezzo
- Progettare anche la classe di ciascun pezzo (a forma di L, a forma di T, a serpente, a forma di I e cubo)
- Progettare infine la classe Tetris che somministra i pezzi, permette al giocatore di muoverli, gestisce lo spazio libero e calcola i punteggi del giocatore