Metodologie di Programmazione

Lezione 26: L’uguaglianza in Java

Lezione 26: Sommario

UNITELMA SAPIENZA
SAPIENZA UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Mutua ricorsione
- Ricorsione di coda
- Uguaglianza in Java:
  - compareTo
  - equals
  - hashCode

Mutua ricorsione

- A volte può verificarsi il caso in cui il metodo a richiami b e b, a sua volta, richiami a
- Questo meccanismo di reciproca chiamata dei metodi è detto di mutua ricorsione
- Ovviamente è sempre necessario stabilire uno o più casi base, onde evitare un ciclo infinito di chiamate tra a e b

Esempio: controlla numeri pari e dispari

- Scrivere una classe che, data una lista di interi, controlli ricorsivamente se ciascun numero in posizione pari sia dispari e ciascun numero in posizione dispari sia pari

```
public class PariDispari
{
    public boolean pariDispari(ArrayList<Integer> l)
    {
        return pariDispari(l, 0);
    }

    private boolean pariDispari(ArrayList<Integer> l, int k)
    {
        if (k == l.size()) return true;
        return (l.get(k) % 2 == 0) && dispariPari(l, k+1);
    }

    private boolean dispariPari(ArrayList<Integer> l, int k)
    {
        if (k == l.size()) return true;
        return (l.get(k) % 2 == 1) && pariDispari(l, k+1);
    }
}
```

Esercizio: Somma e sottrazione di valori

- Scrivere una classe SommaSottrai che, costruita con una lista di interi, espone un metodo ricorsivo sommaSottrai che restituisce la somma dei valori della lista in posizione dispari da cui vengono sottratti i valori in posizione pari
  - Ad esempio, data la lista [2, 5, 3, 7, 11, 1] il metodo restituisce 2-5+3-7+11-1=3

Ricorsione di coda

- Un tipo particolare di ricorsione in cui l'elenco delle chiamate ricorsive può essere rappresentato mediante una sequenza lineare
- Avviene quando la chiamata ricorsiva è l'ultima operazione del metodo e il chiamante non deve attendere l'esecuzione del metodo per effettuare ulteriori operazioni

```
public int cerca(String s, char k, int p)
{
    if (p >= s.length())
        return -1;

    char c = s.charAt(p);
    if (c == k)
        return p;

    return cerca(s, k, p + 1);
}

public int cerca(String s, char k)
{
    return cerca(s, k, 0);
}
```

Esercizio: And tra espressioni in una lista

- Si progetti un'interfaccia Valutabile che espone un metodo valuta() il quale calcola il valore dell'oggetto e restituisce un booleano
- Si progetti quindi una classe ValutaValutabili i cui oggetti sono costruiti con un array di oggetti "valutabili"
- La classe espone un metodo valutaInAnd **ricorsivo di coda** il quale restituisce il valore booleano dell'and tra tutti i valori degli oggetti nell'array
- Analogamente, realizzare un metodo valutaInOr **ricorsivo di coda** che mette in or i valori degli oggetti nell'array

La Torre di Hanoi

- In un tempio dell'estremo oriente alcuni monaci devono spostare una pila di dischi d'oro da un piolo di diamante a un altro
- La pila iniziale comprende 64 dischi, tutti infilati nello stesso piolo dal basso verso l'alto in ordine di grandezza decrescente
- I monaci devono spostare l'intera pila su un altro piolo, rispettando il vincolo di muovere un solo disco per volta e non ponendo mai un disco più grande sopra uno più piccolo
- Sono disponibili 3 pioli e uno può essere usato come supporto temporaneo
- Si realizzi la classe TorreDiHanoi che risolva il problema

La Torre di Hanoi: suggerimenti

- Tenete bene a mente che per risolvere un problema ricorsivamente è necessario:
  1) ridurlo a un problema più piccolo
  2) identificare il caso base
- Posso riformulare il problema di spostare n dischi nel problema in cui:
  - Sposto n-1 dischi dal piolo 1 al piolo 2 (usando il piolo 3 come supporto temporaneo)
  - Sposto l'ultimo disco (il più grande) dal piolo 1 al piolo 3
  - Sposto n-1 dischi dal piolo 2 al piolo 3, usando il piolo 1 come supporto temporaneo

Uguaglianza in Java

- Abbiamo già parlato della differenza tra equals e ==
  - L'operatore == confronta il riferimento (diciamo, l'indirizzo in memoria), quindi è true **se e solo se** si confrontano gli stessi oggetti fisici
  - L'operatore equals confronta la stringa carattere per carattere e restituisce true se le stringhe contengono la stessa sequenza di caratteri
- Tuttavia, l'implementazione di default di equals (ovvero, in Object) è la seguente:

```
public boolean equals(Object obj)
{
return this == obj;
}
```

hashCode e equals

- Analogamente, l'implementazione di hashCode mappa l'indirizzo in memoria dell'oggetto a un intero

- Perché è necessario sovrascrivere equals e/o hashCode?

- **IMPORTANTE:** Se si sovrascrive l'implementazione di equals è necessario sovrascrivere anche hashCode (e viceversa)
  - **Contratto**: se due oggetti sono uguali secondo equals => **devono avere lo stesso hashCode**

Esempio: MyInteger


```
public class MyInteger
{
    private int v;

    public MyInteger(int v) { this.v = v; }
}
```

- **Problema:** new MyInteger(5).equals(new MyInteger(5)) è false!
- Si risolve sovrascrivendo equals e hashCode

Esercizio: hashCode e equals con le collection

- Implementare una classe PunteggioGiocatore i cui oggetti memorizzano al loro interno l'informazione sul punteggio e sul nome del giocatore corrispondente
- Implementare correttamente i metodi hashCode e equals in modo da garantire la corretta implementazione dell'uguaglianza per la classe
- Verificare il comportamento della classe creando istanze di PunteggioGiocatore con gli stessi valori e con valori diversi
  - Aggiungendo riferimenti agli oggetti a una lista
  - Aggiungendo gli stessi riferimenti a un insieme