Laboratorio di Programmazione
Edizione Diurna

Sessione del 16/GIU/2014 - cognomi da "Ce..." a "Fi..."

Avvertenza: Nello svolgimento dell'elaborato è possibile usare qualunque classe delle librerie standard di Java. Non è invece ammesso l'uso delle classi del package `prog` allegato al libro di testo del Prof. Pighizzini e impiegato nella prima parte del corso.

Tema d'esame

Lo scopo è realizzare delle classi che permettano l'implementazione di una struttura tipo quella rappresentata nella Figura 1, detta *albero binario* (semplificato), in cui tutti gli elementi, detti *nodi* (e rappresentati con cerchi) sono dello stesso tipo, l'elemento più in alto (51 nella figura) è detto *radice* dell'albero e ogni elemento è collegato ad al più due elementi sotto di sé. Dato un nodo A, il nodo immediatamente sopra A è detto *genitore* di A e i nodi immediatamente sotto A sono detti *figli* di A (un po' come nell'albero genealogico). Ad esempio 5 è genitore di 23, e 12 e 46 sono figli di 23.
Le classi da realizzare sono le seguenti (dettagli nelle sezioni successive):

- *SimpleNode*: un nodo dell'albero binario, ha al più due figli, non sa se è parte di un albero (non conosce il suo "genitore")
- *CleverNode* (sottoclasse di *SimpleNode*): ha ANCHE un *genitore*, cioè sa se è parte di un albero
- *Main*: classe che contiene il metodo `main()`

Le classi dovranno esporre almeno i metodi specificati nelle sezioni seguenti. Eventuali metodi di servizio possono essere aggiunti a piacimento. Ogni classe (eccetto la Main) deve avere il metodo `toString()` che rappresenti lo stato delle istanze e i costruttori adeguati per gli attributi che vengono dichiarati. Dato che gli attributi devono essere tutti privati, creare opportunamente, e solo dove necessario, i metodi di accesso (`set` e `get`). Si suggerisce, anche dove non segnalato, di utilizzare, se esistenti e se applicabili, le classi parametriche (es. *ArrayList<E>* invece di *ArrayList*). Alcuni controlli di coerenza vengono suggeriti nel testo, potrebbero essercene altri a discrezione. Si consiglia di posporre l'implementazione dei controlli di coerenza, come ultima operazione, dopo aver realizzato un sistema funzionante.
A titolo di test potete usare il sorgente allegato: *Test.java*.

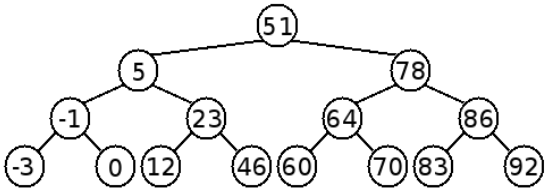

Figure 1: albero binario (fonte Wikipedia)

SimpleNode

Rappresenta un nodo dell'albero, ha due figli (*left* e *right*) e un *content* sottoforma di String.
Gli attributi sono a discrezione, ovviamente serviranno almeno quelli per rappresentare correttamente (scegliendo il tipo opportuno!): *left, right, content.*
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni:

- `public SimpleNode(String content)`: un costruttore che accetta in ingresso la stringa che rappresenterà il contenuto del nodo, deve lanciare un'eccezione nel caso il parametro sia nullo
- `public String getContent()`: restituisce il contenuto testuale del nodo
- `public SimpleNode getLeft()`: restituisce il nodo di sinistra
- `public SimpleNode getRight()`: restituisce il nodo di destra
- `public boolean setLeftNode(SimpleNode n)`: imposta *left* se *left* è libero (null), lancia un'eccezione se n è null, restituisce *true* se ha successo, *false* se ha insuccesso.
- `public boolean setRightNode(SimpleNode n)`: imposta *right* se *right* è libero (null), lancia un'eccezione se n è null, restituisce *true* se ha successo, *false* se ha insuccesso.
- `public String toString()`: non dimenticare! Deve sicuramente ritornare il contenuto ed eventualmente il nome della classe e altre informazioni che riteniate opportune.

CleverNode (extends SimpleNode)

Rappresenta un nodo più "intelligente" dell'albero, ha due figli (*left* e *right*) e un *content* sottoforma di String, ma ha anche un genitore (*parent*), cioè sa se ha qualche nodo "sopra di sé", implementa l'interfaccia Comparable<CleverNode>.
Gli attributi sono a discrezione, ovviamente serviranno almeno quelli per rappresentare correttamente (scegliendo il tipo opportuno!): *left, right, content, parent.*
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni.

- `public CleverNode(String content)`: un costruttore che accetta in ingresso la stringa che rappresenterà il contenuto del nodo, deve lanciare un'eccezione nel caso il parametro sia nullo
- `public CleverNode getParent()`: restituisce *parent* (il nodo superiore) se c'è, null se il nodo non ha genitore (la radice)
- `public void setParent(CleverNode p)`: imposta *parent*, lancia un'eccezione se p è null
- `public boolean setLeftNode(CleverNode n)`: imposta *left* se *left* è libero (null), deve anche impostare il parent di n, lancia un'eccezione se n è null, restituisce *true* se ha successo
- `public boolean setRightNode(CleverNode n)`: imposta *right* se *right* è libero (null), deve anche impostare il parent di n, lancia un'eccezione se n è null, restituisce *true* se ha successo
- `public boolean appendNode(CleverNode n)`: prova ad agganciare (chiamando *setLeftNode* o *setRightNode*) a destra o a sinistra (in modo random) un nuovo nodo, se è possibile (cioè se "c'è posto" - null - a destra o a sinistra), imposta correttamente il parent, lancia un'eccezione se n è null, restituisce *true* se ha successo
- `public boolean sort()`: se i nodi *left* e *right* non sono in ordine (lessicografico sulle stringhe contenute, cfr. metodo *compareTo*) li scambia, restituisce *true* se effettua lo scambio
- `int compareTo(CleverNode o)`: deve confrontare le stringhe contenute (in *this* e o) e restituire un valore basato sul confronto (lessicografico)
- `public String toString()`: non dimenticare!

Main

La classe `Main` contiene il metodo `main`, oltre ad eventuali altri metodi che riterrete opportuni e a quelli specificati più sotto.
A titolo di test potete usare il sorgente allegato: *Test.java*
La classe Main deve essere "lanciata" specificando di seguito su linea di comando il nome del file da cui leggere dei contenuti (stringhe). Il programma deve terminare con un opportuno messaggio nel caso il nome del file non sia stato specificato o nel caso il file specificato non esista.
Nel file (allegato al presente tema: *words.txt*) sono presenti righe di testo separate da *newline*, questo testo dovrà essere usato per la creazione di istanze di `CleverNode`, una riga per ogni istanza, la riga rappresenta il *content* di ogni `CleverNode`, ogni nuovo nodo deve essere "appeso" al precedente (metodo *appendNode*).
Nella classe Main andranno anche implementati i seguenti metodi:

- `public static int lengthLeft(SimpleNode n)`: restituisce il numero di nodi *left* discendenti del nodo n (i.e., nodo figlio sinistro, nodo figlio sinistro del nodo figlio sinistro, ...)
- `public static int lengthRight(SimpleNode n)`: restituisce il numero di nodi *right* discendenti del nodo n (idem come sopra, ma a destra)

Consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti i file *.java* prodotti. NON vanno consegnati i *.class* Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web: http://upload.di.unimi.it
ATTENZIONE!!! NON VERRANNO VALUTATI GLI ELABORATI CON ERRORI DI COMPILAZIONE. UN SINGOLO ERRORE DI COMPILAZIONE INVALIDA **TUTTO** L'ELABORATO.