AbstractNode

We hebben nog niet alle functionaliteit van een node geregeld. Dit zouden we verder in NameNode kunnen doen, maar dan moet deze algemene Node-functionaliteit in andere Node classes worden herhaald. Maak daarom de class AbstractNode aan die Node implementeert, en maak NameNode daarvan een subclass. Verhuis de methoden compareTo(), compareLevelTo(), isLeaf(), getParent() en setParent() naar AbstractNode.

Regel de volgende zaken in AbstractNode:

- Een opgegeven parent node in setParent() mag geen descendant node zijn van de eigen node (this). Maak een private boolean methode isDescendant(E node) die dit controleert.
- Als de opgegeven node wel een geldige parent node is, dan kan die node geen leaf node meer zijn. Mogelijk wordt de vorige parent node dan wel een leaf node. Dat is echter niet zeker: een parent node kan immers meerdere child nodes hebben.
- Maak daarom een variabele childCount in AbstractNode: elke keer dat setParent() correct wordt aangeroepen, wordt childCount van de nieuwe parent node opgehoogd, en die van de vorige parent node verlaagd.

Hoe kunnen we nu eenvoudig de isLeaf() methode binnen AbstractNode implementeren?

Verbeterde sortering

De sortering van nodes is nog niet optimaal. Uiteindijk moet uit de sortering de hiërarchie terug te zien zijn. Te beginnen bij een root node, wordt daarin eerst de eerste hoofdtak opgebouwd, met alle zijtakken, om daarna de volgende hoofdtak te tonen.

De sortering dient dus te starten met een start- of root node op level 0, met daaronder de eerste child node op level 1. Probeer de gewenste volgorde te genereren in public int compareTo(E other), op de volgende manier. Bedenk hierbij steeds wat de return waarde moet zijn als this links of rechts van other terecht moet komen.

- een node zonder parent komt altijd links van een node met een parent te staan.
- een child-node komt altijd rechts van zijn parent te staan
- twee child nodes met dezelfde parent kunnen alfabetisch gesorteerd worden, op basis van toString().
- twee nodes met een gelijk level, die geen geen gezamenlijke parent hebben, hebben ergens een gezamenlijke “grootouder”. Ze behoren dan tot twee verschillende zijtakken, die op basis van toString() t.o.v. elkaar zijn gepositioneerd (zie het vorige punt). De plek van de ene node ten opzichte van de andere node wordt dan dus bepaald door de posities van beide zijtakken t.o.v. elkaar. Om te bepalen tot welke zijtak ze behoren (die op basis van toString() gesorteerd worden), kan de parent van this vergeleken worden met de parent van de ander (dus een recursieve aanroep van compareTo).
- van twee nodes met een ongelijk level die geen parent-child relatie met elkaar hebben dient ook de gezamenlijke voorouder gevonden te worden, om te bepalen tot welke zijtak ze behoren. Vergelijk daartoe de parent van de node met het laagste level (die dus het verst van de gezamenlijke root node afzit), met de andere node (dus weer recursief). Dit levert (uiteindelijk, eventueel na herhaalde recursieve aanroep) een situatie op die aansluit op een van de vorige punten, zodat (uiteindelijk) de juiste positie bekend is.

Bespreek de opdrachten met de docent.