Het pad naar de start node

Een belangrijke functie van de TreeIterator ontbreekt nu nog. De meeste informatie over de nodes ten opzicht van elkaar kunnen we zichtbaar maken door per node te tonen wat het pad is vanaf die node naar de start node. Als bijvoorbeeld node A de parent is van B, en als B de parent is van C en van D, zouden we dat als volgt zichtbaar kunnen maken bij het doorlopen van de tree:

```
A
A/B
A/B/C
A/B/D
```

We hebben dus uit elke node tekstuele informatie nodig (zoals hier de namen A, B, C en D). Daarnaast is een scheidingsteken nodig om de tekstinformatie van elke node te scheiden (hier het teken "/").

Een probleem daarbij is dat we op het niveau van de TreeIterator niet weten uit welke tekst we dat pad op moeten bouwen. Op dat niveau kennen we van elke node alleen de Node eigenschappen. Een voorlopige oplossing kan zijn om daar de toString() methode voor te gebruiken, die altijd aanwezig is. Geef TreeIterator daarom alvast een methode path() die een String retourneert en alleen een separator mee krijgt:

String path(String separator);

In de implementatie in NodeTree zal hierbij dus de toString() van elke node worden gebruikt. Dit komt in een volgende paragraaf aan de orde.

Als we andere tekst dan toString() willen gebruiken, dan wordt het wat ingewikkelder. Pas als we de TreeIterator gaan gebruiken, zoals in TreeApp, wordt bekend welke implementatie van Node we gebruiken, en welke tekst we daaruit willen gebruiken. In het geval van een NameNode kan de naam een geschikte kandidaat zijn, voor andere nodes zou het pad bijvoorbeeld kunnen worden opgebouwd uit unieke identifiers van de nodes.

Het zou dus handig zijn als we bij het gebruik van de TreeIterator aan de applicatie door kunnen geven **hoe** we de gewenste tekst uit de node kunnen halen. Op het hogere niveau van TreeIterator geven we dan alleen aan **dat** we een String uit een Node kunnen halen, maar nog niet hoe. We gaan dit in de volgende stappen uitwerken.

De interface java.util.function.Function

Een interface is geschikt om aan te geven dat een object iets kan, zonder dat de implementatie al bekend is. De nieuwe interface java.util.function.Function is bedoeld om met behulp van een lambda expressie uit een object een ander object te genereren. Deze interface ziet er als volgt uit:

```
interface Function<T,R> {
  R apply(T t);
}
```

Het type R staat hier voor het return type, en het type T voor het type van de parameter.

In onze toepassing komt er in de apply methode een Node binnen (type E), en krijgen we een String waarde terug.

Voeg in TreeIterator daarom de volgende overloaded methode toe:

```
  String path(String separator, Function<E, String> f);
```

De implementatie van path() in NodeTree

In NodeTree moeten we de iterator() methode nu aanvullen met een implementaties van de overloaded methods path().

Voor de eerste methode path() word van elke Node in het pad de toString() gebruikt. Voor de tweede methode path() weten we nog niet welk deel van een Node gebruikt wordt om het pad op te bouwen. We weten wel dat bij het gebruik van path() een object wordt meegegeven van het type Function . Met een aanroep van de apply() methode van het Function object kunnen we de String ophalen.

Tips:

- Maak in de implementatie van path() gebruik van een StringBuilder() object.
- Begin bij de current node
- Voeg de separator toe, en:
    - het resultaat van toString() (in de eerste path() methode), of
    - Het resultaat van de apply(E Node) van het Function object (in de tweede path() methode)
- Herhaal dit voor elk parent.... (zie volgende tip)
- De methode level() kan gebruikt worden in een for loop, om te bepalen hoe vaak dit moet gebeuren
- Bedenk steeds waar in de StringBuilder de informatie moet worden toegevoegd
- Zorg er na de loop voor dat de separator alleen als scheidingsteken tussen nodes wordt gebruikt, dus niet voor de eerste node.

Het gebruik van path() in TreeApp

In TreeApp kan nu de hiërarchie van de tree zichtbaar worden gemaakt met één van de path() methods. Probeer eerst de toString() versie uit, met een zelf gekozen separator, met als mogelijke uitvoer:

a
a/b
a/b/c
a/b/e
a/b/e/f
a/d

Gebruik daarna de Function versie met behulp van een lambda expressie.
Bouw het pad bijvoorbeeld op uit de namen in hoofdletters:

A
A/B
A/B/C
A/B/E
A/B/E/F
A/D

Bespreek de opdrachten met de docent.