Java 8

Een introductie in Lambda's en default methoden

De nieuwe Java versie 8 wordt al lang verwacht en bevat de meest spectaculaire wijzigingen sinds de introductie van generics in Java 5. Na een aantal keren uitstel is de nieuwe versie eind maart definitief uitgeleverd. In dit artikel bespreken we de nieuwe concepten van Java 8 op het gebied van lambda's en nieuwe interface mogelijkheden. Deze nieuwe taalconcepten zijn beschreven in JSR-335.

We laten in dit artikel zien dat we met de nieuwe toevoegingen in de taal met minder regels, beter te onderhouden en begrijpelijke code kunnen schrijven. In dit artikel zullen we ook zien dat de nieuwe taalconcepten bijna tot geen complexiteit toevoegen en dat het gebruik in de praktijk vanzelfsprekend is.
We gaan niet in op de andere JSR's, die zijn toegevoegd aan Java 8. Een compleet overzicht van alle nieuwe features is te vinden op **https://jdk8.java.net/**

Lambda expressies

In 2008 is voor het eerst overwogen om "lambda expressies" (in het kort: lambda's) aan Java toe te voegen. Na een zes jaar lang durend, intensief debat hebben ze nu eindelijk hun weg naar Java gevonden.
Vanuit taalperspectief zijn lambda expressies de meest invloedrijke toevoeging aan Java. Lambda's openen de deuren naar de veelbelovende wereld van functioneel programmeren en introduceren daarmee een paradigma-verschuiving over hoe software met Java geschreven wordt. Syntactisch bekeken zijn lambda's slechts syntactic sugar. Om dit te verduidelijken kijken we naar de methode `listFiles(FileFilter ff)` van `java.io.File`. Om bijvoorbeeld directories te filteren, diende deze methode tot en met Java 7 als volgt gebruikt te worden:

```
new File("/").listFiles(new FileFilter() {
           public boolean accept(File f) {
              return f.isDirectory();
           }
 });
```

Deze constructie is uitermate verbose en moeilijk leesbaar. Met lambda's kan de code voor het instantiëren van de klasse FileFilter terug gebracht worden tot één regel:

```
new File("/").listFiles((File f) -> f.isDirectory());
```

Semantisch zijn bovenstaande voorbeelden identiek. Dit werkt als volgt: als we de signature van de FileFilter interface bestuderen, zien we dat deze interface een enkele, abstracte methode heeft. Zulke interfaces hebben in Java 8 een specifieke naam gekregen, namelijk: *functional interfaces*. Zij worden vanaf Java 8 met de `@FunctionalInterface` annotatie voorzien.
Overal, waar een methode een functional interface als input parameter verwacht, zoals `listFiles(FileFilter ff)`, kan een lambda expressie in plaats van een anonymous inner class of implementatie gebruikt worden. Dit impliceert dat bovenstaande lambda expressie aan een `FileFilter` variabele toegewezen kan worden:

```
FileFilter ff = (File f) -> f.isDirectory();
```

De enige voorwaarde is, dat de signature van de lambda overeenkomt met de signature van de enige method in de functional interface.
In ons voorbeeld gebruikt de lambda expressie File als input parameter en boolean als output, wat overeenkomt met de signature van de enige method in `FileFilter: boolean accept(File f)`.
Matchen op signature in plaats van type is een nieuw concept in Java. Dit concept is zeer krachtig, maar vergt enige gewenning.

Nanne Baars is software developer bij Xebia

Urs Peter is senior consultant bij Xebia