New & Cool

JavaFX is de nieuwe speler binnen de wereld van RIA's (Rich Internet Applications) en desktopapplicaties. Waar Java tot dusver een zwakke speler is geweest op de markt van interactieve desktopapplicaties, is SUN nu hard bezig om met JavaFX het ontwikkelen van interactieve applicaties leven in te blazen. In dit eerste deel beschrijven we een aantal basisprincipes en gaan we in op hoe we een start kunnen maken met JavaFX. In twee volgende artikelen diepen we overige aspecten van JavaFX verder uit.

JavaFX: on the road…

Voordat we met JavaFX kunnen beginnen, zullen we de laatste versie van de JavaFX preview SDK (1.1) moeten downloaden. De benodigde bestanden zijn beschikbaar voor het werken vanaf de command line of via de Netbeans plugin die ondersteuning van JavaFX mogelijk maakt.

Wanneer we de bin directory van de JavaFX SDK toevoegen aan ons systeempad krijgen we de beschikking over een drietal tools:

1. javafxc: De JavaFX compiler;
2. javafx: De benodigde JavaFX runtime;
3. javafxdoc: Voor het genereren van JavaFX documentatie.

Genoemde tools werken op soortgelijke wijze zoals we dat ook kennen uit de Java development kit. Je kunt nu een bestand definiëren met de extensie .fx (bv main.fx) om een eenvoudige JavaFX applicatie te maken. Vervolgens beschrijven we daarin de volgende code zoals we zien in codevoorbeeld 1.

```
println("JavaFX on the road - first steps");
```

Codevoorbeeld 1, eerste stappen met JavaFX.

De 'println' functie is te allen tijde beschikbaar in een JavaFX applicatie en we hoeven hiervoor niets extra te definiëren. We kunnen nu dit JavaFX script compileren door gebruik te maken van 'javafxc' en het vervolgens op te starten met 'javafx'. Codevoorbeeld 2 laat zien hoe we dit voor elkaar krijgen.

```
D:\data\articles\JavaFX  2009\firstproject>javafxc  main.
fx
D:\data\articles\JavaFX 2009\firstproject>javafx main
first steps
```

Codevoorbeeld 2, compileren en opstarten van een JavaFX script.

De JavaFX compiler zal twee bestanden aanmaken waarvan 'main.class' degene is die we echt kunnen opstarten. In dit class bestand is een 'main' methode toegevoegd door de compiler om de script code op te starten. Wanneer we het 'javafx' commando gebruiken dan worden de JavaFX runtime bestanden toegevoegd aan het classpath en vervolgens wordt de 'main' methode in het script bestand aangeroepen.
Momenteel bieden Eclipse en Netbeans ondersteuning voor het ontwikkelen van JavaFX applicaties. Voor dit artikel maak ik gebruik van Netbeans 6.5.

Opzet van een eenvoudige JavaFX applicatie

Om een JavaFX applicatie te bouwen beginnen we met de visuele container, het 'Stage' object (zie Codevoorbeeld 3). Dit is de top container waarin we de rest van de applicatie stukje voor stukje toevoegen. Door het definiëren van accolades kunnen we de gewenste eigenschappen zoals hoogte, breedte en andere attributen van het Stage object instellen voor onze applicatie. Het toevoegen van functionele elementen die nodig zijn voor de applicatie gebeurt door gebruik te maken van het attribuut 'content'. Het voorbeeld hieronder toont ons hoe we met het 'Stage' object een eenvoudige applicatie maken.

```
Stage {
    title: "Simple application"
    width: 250
    height: 80
    scene: Scene {
        content: Text {
            x: 10, y: 30
            content: "Application content"
        }
    }
```

Codevoorbeeld 3: opzet voor een simpele JavaFX applicatie.

In codevoorbeeld 3 zien we hoe door middel van de declaratieve aard van JavaFX script we direct een Stage object instantiëren. Het is even wennen aan de notatie, maar door het object te definiëren en

Ronald van Aken
is werkzaam bij Accenture Technology Solutions en specialiseert zich op het gebied van Java, BPM en SOA oplossingen.

Het is even wennen aan de notatie, maar we krijgen direct een instantie

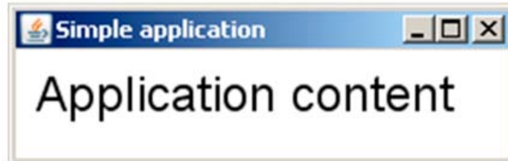
accolades te gebruiken, krijgen we direct een instantie van het desbetreffende object. Een belangrijk element in het Stage object is het attribuut scene. Door middel van dit attribuut kunnen we nieuwe elementen toevoegen aan onze JavaFX applicatie. In dit geval voegen we een Text object toe aan ons Scene object. Via het content attribuut van het Text object voegen we een stukje tekst toe. Het 'content' object kan ook een verzameling van andere objecten bevatten. Dit toon ik later door gebruik te maken van een 'Group' object. De aanpak in het voorgaande voorbeeld kenmerkt de declaratieve stijl die wordt gebruikt in JavaFX applicaties. Om de punten op de i te zetten is het nog nodig om de benodigde packages te definiëren waar de klassen kunnen worden gevonden waaraan we refereren. Codevoorbeeld 4 toont ons hoe we dit met JavaFX voor elkaar krijgen.

```
package javafxexamplenljug1;

import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
```

Codevoorbeeld 4: benodigde packages voor voorbeeld 1.

Het definiëren van de benodigde packages is nagenoeg (en gelukkig) gelijk aan wat we in Java gewend zijn.



Figuur 1: Resultaat van voorbeeld 1.

Met dit voorbeeld kun je zien hoe we door middel van de scripteigenschappen van JavaFX al snel een resultaat kunnen boeken. JavaFX is een scripttaal die wel is opgebouwd uit klassen en objecten die we gebruiken in onze scripts. Echter de visie om het meer scriptgeoriënteerd in tegenstelling tot puur objectgeoriënteerd te laten werken, zorgt ervoor dat we heel snel de beschikbare objecten kunnen combineren om tot een resultaat te komen.

In Codevoorbeeld 3 maken we gebruik van het content attribuut om een ander object toe te voegen. Vaak zul je echter meerdere objecten nodig hebben die samen de applicatie vormen. Wanneer we een groep van objecten willen bundelen tot één entiteit gebruiken we hiervoor een 'Group'. Binnen een groep kun je een verzameling van JavaFX objecten definiëren. Hierdoor kunnen we verschillende objecten plaatsen in ons Scene object. Een voorbeeld hoe we dit doen zie je in codevoorbeeld 5. In dit voorbeeld definiëren we een Circle object. Vervolgens definiëren we een Text object. Binnen het Stage object gebruiken we de Group om de eerdere objecten ("circle" en "text") te bundelen.

```
var circle = Circle{
    radius: 20;
    centerX: 20;
    centerY: 20;
}

var text = Text{
    x: 10
    y: 30
    content: "Application content"
}

Stage {
    width: 250
    height: 80
    scene: Scene {
        content: Group{
            content: [circle, text]
        }
    }
}
```

Codevoorbeeld 5: gebruik van het Group object.

JavaFX profielen

Profielen worden in JavaFX gebruikt om ondersteuning te geven voor platformspecifieke zaken. Hierbij moeten we onder andere denken aan specifieke profielen voor mobiele apparaten, desktops en internetbrowsers. De API die beschikbaar is binnen JavaFX is opgedeeld in profielen. Het gebruik van het 'common' profiel stelt ontwikkelaars bijvoorbeeld in staat om applicaties te ontwikkelen voor alle beschikbare platformen, waar het gebruik van het 'desktop' profiel specifieke classes levert voor dit platform en die niet noodzakelijk compatible zijn met het gebruik ervan op de andere platformen. Hoe de API is verdeeld over de verschillende profielen is het beste te zien op http://java.sun.com/javafx/1.1/docs/api/. Wanneer we als JavaFX ontwikkelaar aan de slag gaan met de API zul je dus zelf moeten opletten dat je alleen klassen gebruikt die in het profiel zitten dat bestemd is voor het doelplatform. Je kunt dit niet codematig afdwingen en dit vergt dus kennis en discipline. Momenteel zijn de volgende profielen gedefinieerd voor JavaFX:

Profiel naam	Doel
Common	Algemene functionaliteit, compatible voor alle platformen
Desktop	Specifieke objecten die het mogelijk maken om de desktop applicaties mee te realiseren
Mobile	Ontwikkelen van JavaFX applicaties voor mobiele apparaten

JavaFX script vs classes

Tot zover hebben we alle code in een enkel JavaFX bestand geplaatst. Het is gelukkig ook mogelijk om JavaFX classes te maken en deze als herbruikbare entiteiten te gebruiken in applicaties. Dit geeft ons de basis om objectgeoriënteerde ontwerpprincipes toe te passen binnen JavaFX applicaties. Een voorbeeld van een eenvoudige JavaFX klasse zie je in codevoorbeeld 6.

```
public class Book {
```

```
    var ISDN: String;
    var title: String;
    var author: String
}
```

Codevoorbeeld 6: eenvoudige JavaFX klasse.

De klasse 'Book' is een eenvoudig voorbeeld waarin we alleen een aantal attributen toekennen die bij de klasse horen. Het sleutelwoord 'var' is nieuw en wordt gebruikt in JavaFX om variabelen te declareren. Zoals we nu de attributen definiëren komen ze in de standaard scope terecht en zijn alleen binnen deze klasse te gebruiken (private) en niet aan te roepen door andere objecten of scipts. Wanneer we bepaalde attributen bereikbaar willen maken dan moeten we ze 'public' of 'protected' definiëren.

```
public class Book {
    public var ISDN: String;
    public var title: String;
    public var author: String;
    protected var id;
}
```

Codevoorbeeld 7: definiëren van scope op attributen.

In codevoorbeeld 7 zien we hoe door middel van van de sleutelwoorden 'public' de attributen van onze klasse publiekelijke beschikbaar maken. Daarnaast is er nog een extra attribuut 'id' toegevoegd zonder type declaratie. Deze zal standaard hierdoor van het type 'Object' zijn.
Het definiëren van methoden voor de klassen gebeurt door middel van het sleutelwoord 'function'. In het onderstaande voorbeeld zie je hoe de klasse uitbreiden met een simpele methode.

```
public class Book {
    public var title: String;
    public var author: String;

    function getBookTitle(): String{
        return "{title} written by {author}";
    }
}
```

Codevoorbeeld 8: definiëren van methodes.

Om het gewenste resultaat te creëren retourneren we een eenvoudige String. Deze is opgebouwd uit een stukje statische tekst en de waarden van de twee attributen. Door gebruik te maken van de accolades in de String kun je refereren aan de attributen. Dit is een voorbeeld van een tekstuele expressies. Het onderwerp van expressies in JavaFX zal ik in een volgend artikel meer in detail behandelen aangezien expressies een belangrijk element vormen in de JavaFX taal.
Let op dat bij het definiëren van functions dezelfde scope regels gelden, en dus betekend dat de methode die in codevoorbeeld 8 in dit geval dus automatisch 'private' wordt.
Het gebruik van polymorphisme is beschikbaar in JavaFX, wel moet er expliciet worden gedefinieerd dat de methode in de base class verder uitbreidt. We doen dit met het sleutelwoord 'override', een voorbeeld hiervan zie je in Codevoorbeeld 9. Opvallend is hier dat als we de methode in de bovenliggende class willen aanroepen we gebruik moeten maken van de syntax die we in Java gebruiken voor het aanroepen van statische methoden en er als volgt uit zal zien 'Employee.getName();'. Binnen JavaFX hebben we daarom het sleutelwoord 'super', niet meer nodig. Nota bene het gebruik van 'static' wordt niet ondersteund in JavaFX.

```
public abstract class Employee extends Serializable {
    public var employeeNumber: Number;
    public var name: String;

    protected function getName(): String{
                  return  "Employee:  {name},  number:
{employeeNumber}";
    }
}

public class SalesEmployee extends Employee{
    public var salary: Number;

    override function getName():String{
      return "The sales employee {Employee.getName()} has
a salary of {salary} ";
    }
}
```

Codevoorbeeld 9: polymorphisme in JavaFX.

Zoals we zien, lijkt JavaFX script qua scope van objectoriëntatie veel op Java, maar wel met aanzienlijke verschillen in de syntax ervan. Een belangrijk punt dat echt anders is het feit dat JavaFX het definiëren van interfaces niet ondersteund, wel kun je een bestaande Java interface implementeren. Hiermee wordt het mogelijk om een JavaFX instantie als argument mee te geven aan bestaande Java code dat als argumenttype een bepaalde interface verwacht! Hoe je een bestaande Java-interface implementeert zie je in codevoorbeeld 10.

```
import java.io.Serializable;

public class Employee extends Serializable {

}
```

Codevoorbeeld 10: JavaFX class die een bestaande Java interface implementeert.

Wat ons als Java adept opvalt, is het gebruik van het sleutelwoord 'extends' in plaats van 'implements'. JavaFX heeft geen ondersteuning voor het laatste. We komen hiermee direct op het volgende opvallende aspect. Dat betekent dat JavaFX ondersteuning biedt aan het overerven van meerdere classes. Persoonlijk moest ik hier even aan wennen en vroeg me af waarom Sun opeens hier zo van koers is gewijzigd ten opzichte van de Java-taal. Het antwoord hierop is dat door dit concept het direct gemakkelijker wordt voor JavaFX script ontwikkelaars om direct verschillende classes met elkaar te combineren. Zonder dat zij zich al te druk hoeven

Wat opvalt is het gebruik van het sleutelwoord 'extends' in plaats van 'implements'

te maken over het feit of ze nu een interface of een abstracte class moeten gebruiken/definiëren.

Omdat JavaFX de visie hanteert om een eenvoudige scripttaal te zijn, is ook het concept van encapsulatie anders opgelost. Waar we in Java hebben geleerd om attributen netjes via 'get' en 'set' methodes te initialiseren en op te vragen is dit niet nodig in JavaFX. JavaFX levert ons wel een voorziening om attributen te beveiligen door zogenoemde triggers te definiëren die afgaan wanneer een attribuut wordt aangepast. Dit wordt gedaan door de 'on replace' syntax. Hiermee kunnen we een stuk validatiecode schrijven dat de nieuwe waarde controleert, voordat die wordt toegekend aan het attribuut. Een voorbeeld hiervan is te zien in codevoorbeeld 11

JavaFX hanteert de visie van een eenvoudige scripttaal

```
public var salary: Number on replace oldValue {
        if (salary <= 0) {
            salary = oldValue;
        }
    }
```

Codevoorbeeld 11: gebruik van on replace trigger voor een attribuut.

Constructors zijn in JavaFX niet beschikbaar, maar u kunt wel een methode 'postinit' definiëren waarin initialisatie code kan worden geplaatst.

Het gebruik van bestaande Java-code in JavaFX applicaties

Het aanroepen en combineren van normale Java-code in JavaFX applicaties kan zonder problemen worden gedaan. Simpelweg door de juiste classes te importeren en te zorgen dat de benodigde JAR-bestanden op het classpath kunnen worden gevonden, zijn we in staat om bestaande Java code aan te roepen.

```
import java.lang.System;

System.out.println("On the road");
```

Codevoorbeeld 12: aanroepen van Java code vanuit JavaFX

Codevoorbeeld 12 toont ons hoe we door het importeren van de juiste packages vanuit een FX bestand Java-code kunnen aanroepen zoals we dat gewend zijn. Dit zorgt ervoor dat we in principe alle Java-code kunnen gebruiken wanneer dit nodig is. Bijvoorbeeld het hergebruik van Java loggers en andere hulpbibliotheken kunnen ons helpen bij de ontwikkeling van JavaFX applicaties. Ik zie hier vooral een voordeel dat we bedrijfslogica in normale Java-code kunnen encapsuleren en dit kunnen hergebruiken in JavaFX code. In codevoorbeeld 13 zie je hoe we een Java code aanroepen .

```
import com.example.client.model.CustomerManager;
// Search for customers which match the given value in
their name

public function searchCustomers(name: String){
    var manager = new CustomerManager();
    customers = manager.findCustomersByName(name);
}
```

Codevoorbeeld 13: gebruik van Java in JavaFX.

Databinding in JavaFX

Het principe van databinding dat is toegevoegd aan het JavaFX platform zorgt ervoor, dat wanneer de waarde van x wijzigt dit automatisch kan worden gesynchroniseerd met een andere variabele of attribuut van een object. Databinding kunnen we realiseren door het gebruik van het sleutelwoord 'bind' en codevoorbeeld 14 toont ons hoe we een visueel tekstveld initialiseren met de waarde van het attribuut 'name' van een Customer object.

```
var customer = Customer{
  name: "Ronald"
};

var nameTextBox = SwingTextField{
    text: bind customer.name;
}
```

Codevoorbeeld 14: Synchroniseren van GUI met het model.

Het omgekeerde is ook mogelijk met het sleutelwoord 'with inverse'. Codevoorbeeld 15 toont ons hoe we de waarde van een tekstobject, wanneer dit wijzigt, direct wordt gesynchroniseerd met het attribuut 'name' van het object 'customer' en vice versa. Het principe van databinding is een erg krachtig element dat ervoor zorgt dat we op een eenvoudige manier objecten aan elkaar kunnen koppelen zonder dat we zelf hiervoor synchronisatie (glue) code hoeven te schrijven en zorgt voor compacte code.

```
var nameTextBox = SwingTextField{
    text: bind customer.name;
}
```

Codevoorbeeld 15: tweeweg synchronisatie tussen model en GUI.

Tot slot

In dit eerste deel van een serie over Java FX heb ik een eerste indruk gegeven hoe we een begin kunnen maken met JavaFX en daarbij vooral gekeken naar de objectgeoriënteerde aspecten van JavaFX. Wat mij in het bijzonder opvalt, is dat wanneer we eenmaal gewend zijn aan de afwijkende object syntax we heel snel op basis van onze bestaande Java-ervaring JavaFX classes kunnen maken en gebruiken in scripts. In de artikelen die volgen zullen we veel meer ingaan op de specifieke aspecten van JavaFX en de details bespreken.
We zullen onder andere laten zien hoe krachtige concepten als databinding, animaties en andere visuele aspecten kunnen worden gebruikt om heel snel mooie resultaten te boeken zonder dat we daar heel veel code voor nodig hebben en de script syntax en andere functionaliteiten die JavaFX ons biedt optimaal gebruiken. «