New & Cool

In het eerste deel van dit tweeluik hebben we Java FX, de nieuwe oplossing van Sun voor het bouwen van RIA's, geïntroduceerd.. In dit tweede deel gaan we uitgebreid in op de script aspecten van JavaFX en bespreken we welke grafische mogelijkheden JavaFX ons biedt.

JavaFX: On the road...

Scripting & graphics in JavaFX

JavaFX is voortgekomen uit de scripttaal F3 (Form Follows Function), F3 is oorspronkelijk ontworpen door Chris Oliver. Nadat SUN het bedrijf waarvoor Chris Oliver werkzaam was overnam, adopteerde het bedrijf F3 en wordt het nu in de markt gezet als JavaFX. Dat JavaFX een belangrijk onderdeel is van de totale strategie voor het Java-platform werd benadrukt op de laatste JavaOne conferentie. De intentie bij het ontwerp van deze taal was om een gemakkelijke taal te ontwerpen waarmee ontwikkelaars snel GUI-applicaties konden realiseren. F3 was initieel opgezet als interpreteerbare code, wat wil zeggen dat het een code is die pas op moment van uitvoer wordt geëvalueerd en omgezet naar machinecode. Later heeft men de keuze gemaakt om de code te compileren naar uitvoerbare bytecode die op elke JVM kan worden uitgevoerd, echter het is ook nog steeds mogelijk om op runtime niveau JavaFX script te interpreteren.

De syntax en scripteigenschappen

JavaFX is een scripttaal die in de basis veel op Java lijkt, maar die ook veel extras biedt ten opzichte van Java. Wanneer je met JavaFX begint is het daarom goed om je open te stellen voor een nieuwe taal en niet teveel op bestaande Java-ervaring te leunen. Je zult snel merken dat JavaFX een hoop dingen net iets anders noemt en extra sleutelwoorden beschikbaar heeft.

Persoonlijk viel mij op dat er veel aandacht is besteed aan het gebruik van expressies en de mogelijkheid van het gebruik van verkorte notaties, wat ervoor zorgt dat we met weinig code heel snel resultaten kunnen boeken. Een voorbeeld van hoe verkorte notaties kunnen worden toegepast wordt later in dit artikel in de paragraaf 'Animatie in JavaFX' behandeld. Eerst zullen we een aantal basiseigenschappen van JavaFX-scripting bespreken.

Een aantal belangrijke sleutelwoorden in JavaFX zijn:

Keyword	Betekenis	
Var	Definitie van een variabele	var integer i = 0;
Function	Methode met resultaat	function createCustomer(): Customer{};
Def	Definitie van een constante waarde	def String applicationtitle = "My application";

Tabel 1.

Een voorbeeld van het gebruik van deze sleutelwoorden zie je in Codevoorbeeld 1.

```
var outcome: Number;
def radius = 30;

function d(): Number{
    return outcome * radius;
}
```

Codevoorbeeld 1: gebruik van de basis keywords.

Bij JavaFX is het belangrijk om te realiseren dat alles als een expressie wordt behandeld. Dit zorgt ervoor dat we zeer krachtige statements kunnen maken met relatief weinig code. Codevoorbeeld 2 toont het gebruik van expressies tijdens het declareren van variabelen.

```
// Eenvoudige definitie van een variable met een
tekstuele waarde
var title = "Example";
/** Definitie van een instantie Customer met daarin direct
initialisatie van het  attribuut "name" */
var customer = Customer{ name: "Ronald" };
/** Initialisatie doormiddel van toekenning via de waarde
van een attribuut van een bestaand object */
var length = title.length();
//  Initialisatie  doormiddel  van  een  sommatie  van
verschillende variabelen
var width = {length + defaultWidth};
// Initialisatie doormiddel van concattenatie van een
tweetal Strings
var name = {title}{" with the length of "}{length};
```

Codevoorbeeld 2: gebruik van expressies bij declaraties.

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

JavaFX biedt als taal meer aspecten dan we al kennen vanuit Java.

Het concept van arrays keert in JavaFX in de vorm van 'sequences' terug. Sequences kunnen worden gebruikt om verzamelingen te definiëren. Het gebruik van blokhaken definieert een sequence en, net als in Java, is het alleen toegestaan om types van hetzelfde soort in een sequence te plaatsen.
Een mooie bijkomstigheid is dat expressies gebruikt kunnen worden in combinatie met sequences. Een voorbeeld hiervan is het definiëren van een sequence welke een subset is van een eerder gedefinieerde sequence zoals te zien is in Codevoorbeeld 3. Het voorbeeld resulteert in een subset van de verzameling 'subsetNumbers' waarin alleen waarden worden geplaatst die groter zijn dan 3. De expressie gebruikt 'n' als variabele om te evalueren; vervolgens kunnen we de conditie specificeren waaraan moet worden voldaan. Let op dat het '|' teken hier dus niet de semantiek heeft van 'or' zoals we dat in Java kennen.

```
// standaard sequeence declaratie
var number = [1,2,3];
//Voeg alleen waarden toe aan de subset die groter zijn
dan 3
var subsetNumbers = number [n | n>3];
// controleer of i in range van 1 tot 10 valt; indien
dat zo is vermenigvuldig dan met zichzelf
var squares = for (i in [1..10]) i*i;
```

Codevoorbeeld 3: definitie van sequences.

Dezelfde principes kunnen worden gehanteerd bij niet-primitieve waarden. Je kunt ze bijvoorbeeld ook gebruiken wanneer je een sequence hebt met daarin objecten en waarbij criteria worden gedefinieerd voor één van de attributen van het object. Een voorbeeld hiervan zien we in Codevoorbeeld 4.

```
public class Customer {
    public var name: String;
    public var address: String;
    public var age: Integer;
    override function toString():String{
        return "{name} and I am {age} years old";
    }
}

var customers = [Customer{name: "Ronald", age: 25(Ja
joh? ;) )},
                 Customer{name: "Klaas", age: 26},
                 Customer{name: "Jenny", age: 29}];

var subSetCustomers = customers[ c | c.age >= 26];

function run():void{
    println(subSetCustomers);
} //* prints [ Klaas and I am 26 years old, Jenny and I
am 29 years old ]*/
```

Codevoorbeeld 4: Voorbeeld van het vullen van een sequence met objecten op basis van een expressie.

Bij het gebruik van sequences zijn een aantal keywords toegevoegd die het mogelijk maken om de sequence te manipuleren. Om de inhoud te manipuleren hebben we 'insert' en een 'delete' statement ter beschikking.
De volgende mogelijkheden zijn beschikbaar:

Keyword	Effect
Insert <value> into <sequence>	Voegt een waarde aan het eind van de sequence toe.
Insert <value> into before <sequence>-(index|expression)	Voegt een waarde toe waar de positie wordt bepaald op basis van index of een expressie
Insert <value> into after <sequence>(index|expression)	Voegt een waarde toe waar de positie wordt bepaald op basis van index of een expressie
Delete <sequence>	Verwijderd alle waardes in de sequence
Delete <sequence>(index|expression)	Verwijderd alle warden op basis van index of een expressie

Tabel 2.

Het gebruik van deze keywords wordt in Codevoorbeeld 5 verduidelijkt.

```
var numbers = [1,2,3,4,5,6,7,8,9];

insert 13 into numbers; // Results
[1,2,3,4,5,6,7,8,9,13]
insert 13 before numbers[0]; // Results
[13,1,2,3,4,5,6,7,8,9,13]
insert 14 after numbers[0]; //Results [ 13, 14, 1, 2, 3,
4, 5, 6, 7, 8, 9, 13]
delete numbers[0]; // Results [14,1,2,3,4,5,6,7,8,9,13]
delete numbers; // Results []
```

Codevoorbeeld 5: gebruik van insert en delete bij sequences.

Het gebruik van expressies bij de initialisatie van de sequence geeft ons de mogelijkheid om de sequence direct naar wens te kunnen initialiseren. In Codevoorbeeld 6 zien we de kracht van expressies in JavaFX dan ook goed naar voren komen.

```
var number = [1,2,3];
var total = {
    var sum:Integer = 0;
    for (n in number){
        sum += n;
        }
    sum;
}

var text = Text { content: "the sum is {total}" };
```

Codevoorbeeld 6: gebruik van expressies bij initialisatie.

We zien dat variabele 'total' zijn waarde krijgt vanuit een expressie die we uitschrijven in een codeblok. We doorlopen de waarden van de sequence 'numbers' en tellen deze op in de variabele 'sum'. Vervolgens geven we aan dat de waarde van 'sum" moet worden toegekend aan "total'. Het object 'Text' dat we in bovenstaand voorbeeld gebruiken, dient om een tekstuele waarde in de applicatie weer te geven en is het equivalent van een label in visuele GUI toepassingen gemaakt met Swing, Delphi en of C#.

Met het beschrijven van bovenstaande aspecten van de scripttaal hoop ik een beeld te hebben gegeven

dat JavaFX als taal veel andere aspecten biedt dan we kennen vanuit Java.

Grafische objecten in JavaFX

Voordat we iets kunnen zeggen over animaties moeten we eerst de grafische architectuur van JavaFX uiteenzetten. In JavaFX worden grafische elementen geplaatst binnen een 'Scene' object. Dit object is de container waarmee we een grafische hiërarchie binnen een JavaFX-applicatie kunnen opbouwen. Het Scene object bestaat uit branches en nodes, waar een branche 0 tot n nodes kan hebben en een node geen verdere vertakkingen heeft. Grafische manipulaties in een JavaFX-applicatie kunnen uitgevoerd worden op nodes binnen een branche. Het opbouwen van de nodes in een applicatie is weergegeven in Codevoorbeeld 6. Dit voorbeeld toont hoe een cirkel en een tekst node aangemaakt worden en hoe we deze in het scene object plaatsen van onze JavaFX-applicatie. Het Stage object is het beste te vergelijken met de root container van een applicatie (SwingContainer in Java). In dit specifieke geval bevat de Scene (de branche) een tweetal nodes (Text & Circle).

```
Stage {
    width: 250
    height: 80
    scene: Scene { // scene is een branch met twee
nodes
         content: [
             Text { // Node 1
                  font : Font {
                      size : 24 }
                  x: 10, y: 30
                  content: "Application content"} ,
             Circle { // Node 2
                  radius: 40;
                  fill: Color.AQUA; }]
    }
```

Codevoorbeeld 7: definiëren van grafische hiërarchie in het scene object.

Binnen JavaFX is een aparte package gedefinieerd met daarin verschillende grafische vormen die we out of the box kunnen gebruiken; daarnaast bestaat de mogelijkheid om polygonen te gebruiken waarmee we vervolgens zelf nieuwe vormen kunnen definiëren. Door middel van de standaard beschikbare functionaliteit zijn we in staat om gemakkelijk eenvoudige vormen in onze applicatie te gebruiken. Daarnaast kunnen we ook images en video bestanden in beperkte formaten integreren in JavaFX. In dit artikel wordt echter alleen gebruikgemaakt van de standaardvormen. Hoe we een vorm kunnen tekenen hebben we al gezien in het gebruik van het Circle object in Codevoorbeeld 7, het resultaat hiervan is te zien in afbeelding 1.

Afbeelding 1: resultaat van code voorbeeld 7.

Animatie in JavaFX

Het toevoegen van code die visuele objecten kan animeren is eenvoudig te doen door middel van de beschikbare 'animation' package. Wanneer we met 2D animaties aan de slag gaan, is het voornaamste om te definiëren hoe de animatie zich manifesteert binnen de dimensietijd. Hiervoor is het TimeLine object beschikbaar. Dit object stelt ons in staat om objecten op basis van tijd te manipuleren. Het TimeLine object bestaat uit een KeyFrames object dat op zijn beurt een of meerdere KeyFrame objecten bevat. Het KeyFrame object geeft een definitie van hoe lang iets moet duren en welke manipulatie uitgevoerd moet worden.

Binnen het KeyFrame object moeten we een tweetal eigenschappen goed instellen. Ten eerste bepalen we het moment waarop een KeyFrame actief moet zijn in tijd, dit doen we met het attribuut 'time'. De notatie die we voor dit attribuut gebruiken is een specifiek gebruik van de JavaFX Duration klasse welke wordt gebruikt om tijdindicaties te specificeren.
Vervolgens moeten we ook definiëren wat de waarde moet zijn van een specifieke variabele op dat moment in tijd. Codevoorbeeld 8 laat zien hoe we een TimeFrame gebruiken om een textlabel 360 graden te laten roteren. De waarde van de hoek waarop de tekst wordt weergegeven, wordt tussen de twee tijdsmomenten opgehoogd door het TimeFrame object. Door de waarde 'angle' via databinding te verbinden aan de eigenschap 'rotate' van het tekstlabel wordt de hoek waarin de tekst is weergegeven automatisch aangepast.

```
var angle = 0;
var text = Text{
    font : Font { size : 24 }
    x: 10
    y: 30
    content: "demo";
    rotate: bind angle;
}

Timeline {
  repeatCount: Timeline.INDEFINITE
  autoReverse: true
  keyFrames: [
         KeyFrame{time: 0s; values: [angle=>0]  }, //
startpoint
         KeyFrame{time: 6s; values: [angle=>360]  }  //
endpoint in timeframe
  ]
}.play();
```

Codevoorbeeld 8: definiëren van een KeyFrame voor het animeren van objecten.

In JavaFX hoeven we het KeyFrame object niet expliciet te definiëren. We kunnen gebruik maken van een verkorte notatietechniek om de code compact en eenvoudiger te houden. In Codevoorbeeld

In beperkte formaten kunnen we images en video's in JavaFX integreren.

9 zie je hoe we een animatie toevoegen aan onze simpele voorbeeld applicatie.

```
var circle = Circle{
    radius: 20;
    centerX: bind x;
    centerY: 20;
    fill: LinearGradient {
    startX: 0.0, startY: 0.0, endX: 0.0, endY: 1.0,
proportional: true
        stops: [
            Stop {offset: 0.0 color: Color.BEIGE},
            Stop {offset: 1.0 color: Color.BROWN}
        ]//Stops
    }
}

var x: Number;

Timeline {
  keyFrames: [
         at (0s) {x => 0.0},
```

Codevoorbeeld 9: toevoegen van animatie.

Het voorbeeld laat de cirkel horizontaal bewegen. Dit wordt gerealiseerd door de toevoeging van de TimeLine. Binnen de TimeLine definiëren we de KeyFrames met daarin een tweetal KeyFrame objecten. Dit is in het voorbeeld niet direct zichtbaar, omdat we gebruikmaken van de verkorte notatie die ervoor zorgt dat het KeyFrame object impliciet aanwezig is. Voor elk KeyFrame moeten we een starttijd opgeven en in ons geval geven we voor het eerste KeyFrame een starttijd van 0 seconden op. Vervolgens geven we aan dat op tijdselement 0s de variabele x de waarde 0.0 heeft. In de volgende KeyFrame definiëren we dat op tijdsmoment 4s de variabele x de waarde 158.0 moet hebben en door gebruik van het sleutelwoord 'tween' het verloop naar deze waarde lineair geïnterpoleerd moet worden. Nu rest ons alleen nog de vraag hoe we de variabele x koppelen aan het attribuut centerX (die de positie op de x as bepaalt) van ons cirkelobject, zodat de cirkel een nieuwe positie krijgt. Dit doen we door de volgende code te gebruiken: 'centerX: bind x'. Hierdoor wordt de waarde van attribuut centerX automatisch gesynchroniseerd met de variabele x wanneer deze van waarde wijzigt.

In de behandelde voorbeelden was de animatie erg eenvoudig: het object maakte alleen een horizontale beweging. Meer geavanceerde animaties kunnen we definiëren door gebruik te maken van een 'Path' en 'PathTransistion' object. Dit laatste object zorgt ervoor dat je een grafisch object over een pad van punten kunt laten bewegen. Een Path object definieert een bepaald pad wat een specifiek object moet volgen. In ons volgende voorbeeld definiëren we een eenvoudige lijn die we tekenen door middel van het Path object. Binnen het Path object gebruiken we elements om het pad te definiëren, in dit geval maken we een eenvoudige cirkel. De code die we hiervoor nodig hebben zie je in Codevoorbeeld 10.

```
def path = Path{
   stroke: Color.BLACK
   strokeWidth: 4
   elements: [
      MoveTo  {x: 100 y: 100 },
      ArcTo { x: 250  y: 250  radiusX: 50  radiusY: 50
},
      MoveTo  {x: 250  y: 250 },
      ArcTo { x: 100  y: 100  radiusX: 50  radiusY: 50
}
  ]
}

function run(): Void {
    Stage {
           scene: Scene {
            width: 400
            height: 300
            content: [ path,circle ]
        }
    }
    animation.play();
```

Codevoorbeeld 10: gebruik van een Path object.

Vervolgens kunnen we een PathTransistion object gebruiken om een ander object het gedefinieerde pad te laten volgen. Het PathTransistion.path attribuut kun je gemakkelijk initialiseren door het aanroepen van de utillity klasse: 'AnimationPath. createFromPath(path)'. Binnen de 'run' functie wordt de 'animation.play()"' aangeroepen, de code van dit object (PathTransistion) is uitgeschreven in Codevoorbeeld 11.

```
def circle = Circle {
    radius: 5
    fill: Color.RED
};

def animation = PathTransition {
    node: circle;
    path: AnimationPath.createFromPath(path)
    orientation: OrientationType.ORTHOGONAL_TO_TANGENT
    interpolate: Interpolator.LINEAR
    duration: 5s
    repeatCount: Timeline.INDEFINITE
};
```

Codevoorbeeld 11: gebruik van PathTransistion object.

Het gebruik van grafische objecten in JavaFX geeft ons elementaire bouwstenen waarmee we snel, mooie, grafische applicaties kunnen maken. Vooral door het feit dat het relatief eenvoudig is om in de code events aan de objecten te koppelen kunnen we al snel grafische objecten interactief maken. We zouden bijvoorbeeld een rolodex agenda kunnen maken met bewegende bladeren voor elke letter in het alfabet. Een mogelijke oplossing om dit te realiseren is door een rechthoek voor de bladzijde te definiëren en daarop alle contact gegevens te plaatsen. De rechthoek zal dan zodra een letter wordt geselecteerd worden bijgewerkt met de benodigde gegevens om vervolgens een rotatie te maken om zijn rechter- of linkerzijde, waarna vervolgens de gegevens zichtbaar zijn voor de gebruiker. Voorbeelden die soortgelijk gedrag manifesteren worden meegeleverd met de JavaFX SDK in Netbeans.

Door het gebruik van grafische objecten kunnen we snel mooie grafische applicaties maken.

Swing & JavaFX

Momenteel zijn er geen specifieke objecten om user input af te handelen (denk hierbij aan formulieren etc). In de JavaFX API is alleen een TextBox en een Text (label) aanwezig waarmee het invoeren van eenvoudige user input in een JavaFX-applicatie mogelijk wordt. Het is wel mogelijk om een Swing container in JavaFX te gebruiken, echter komen we dan terecht in de complexiteit waar Swing berucht om is. Maar als we complexe gebruikersinteractie willen, is dit op dit moment de enige optie. Het wachten is op SUN om echte JavaFX controls toe te voegen aan het platform, zodat deze beter aansluiten bij de overige grafische objecten van de JavaFX API als vervanging voor het gebruik van de Swing API.

Voor dit voorbeeld hebben we een Swing klasse ontwikkeld die er uitziet zoals in Afbeelding 2 is te zien.

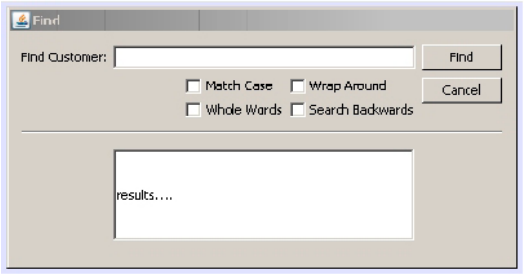
De volgende stap is om deze klasse te laden in het attribuut 'content' van het 'Stage' object van onze JavaFX-applicatie. Hier ontsaat echter een probleem, omdat de Swing klasse hiërarchie niet zomaar in JavaFX kan worden gebruikt. Wat er moet gebeuren, is dat we het Swing object in een container plaatsen die overerft van de klasse 'javafx.scene.Node'. Dit realiseren we door de volgende statische methode aan te roepen: 'SwingComponent.wrap(findDialog);', waar de variabele findDialog een instantie is van de Swing klasse. De volledige code die nodig is om de dialoog in een JavaFX-applicatie te gebruiken zie je in Codevoorbeeld 12.

```
// Instantiate the Swing component
var findDialog = new Find();
// Wrap swing component in a compatible container for
scene.content
var findDialogComp = SwingComponent.wrap(findDialog);

function run():void {
    findDialogComp.clip = Rectangle {
            width: findDialog.getWidth()
            height: findDialog.getHeight()
            };
    findDialog.setVisible(true);
    Stage {
        style: StageStyle.TRANSPARENT
        width: findDialog.getWidth()
        height: findDialog.getHeight()
        scene: Scene {
                content:[ findDialogComp ]
            }
    }
}
```

Codevoorbeeld 12: Gebruik van bestaande Swing code in een JavaFX applicatie.

Er is ook een speciale extensie in de package hiërarchie van JavaFX om ondersteuning te bieden voor Swing componenten zonder het wrappen zoals dat hierboven gebeurt. Helaas zijn er nog geen WYSIWYG tools die het visueel ontwikkelen met



Afbeelding 2.

deze Swing klasses mogelijk maakt. Daarom heb ik ook in dit voorbeeld gekozen om een normale Java Swing klasse te gebruiken.

Tot slot

Naar aanleiding van dit deel van het artikel over JavaFX hoop ik, dat u als lezer kunt concluderen dat het definiëren en manipuleren van grafische objecten gemakkelijk en snel te realiseren is. Met name de script eigenschappen van JavaFX maken het mogelijk dat je snel en efficiënt de objecten kunt manipuleren. Het werken met deze scripttaal kan net iets meer gemak leveren dan de meer strikte Java-syntax; dit geldt zeker voor mensen die niet eerder hebben gewerkt met Java.

De diverse toevoegingen van verkorte notaties zorgen dat de code compact en meer overzichtelijk blijft. Het toepassen van grafische objecten in een JavaFX-applicatie is wat mij betreft laagdrempelig, al is het wel zo dat ik hier soms visuele ondersteuning mis. Ik vraag mij dan ook af of het voor ontwikkelaars niet gemakkelijker kan worden gemaakt. Vooral wanneer we echte complexe gebruikersinteractie gaan ontwikkelen kunnen WYSIWYG-tools veel toegevoegde waarde hebben. Deze tools helpen ons met het genereren van UI code en daardoor het behouden van de focus op het ontwikkelen van de uiteindelijke klantoplossing. Hier hebben concurrenten zoals Adobe zeker een voorsprong! In dit artikel maakten we vooralsnog gebruik van technieken om zelf grafische objecten aan te maken, in het volgende deel zullen we zien hoe de visie van JavaFX erin voorziet hoe grafische ontwerpers en software developers hand in hand kunnen samenwerken en de kracht van JavaFX samen optimaal kunnen benutten. «

Het wachten is op Sun om echte JavaFX controls toe te voegen aan het platform.

Referenties

- http://blogs.sun.com/chrisoliver/category/F3
- http://javafx.com/docs/articles/media/format.jsp
- http://javafx.com/learn/