

Umstiegsanleitung für JavaFX Script auf die Finalversion 1.0

Version 2, Ralph Steyer, 29.12.2008



Videotraining JavaFX

Ralph Steyer

<http://www.video2brain.com/de/products-223.htm>

Addison-Wesley/Video2Brain

ISBN 978-3-8273-6144-8

erschienen 04.2008



Buch JavaFX

Ralph Steyer

[http://www.addison-wesley.de/main/main.asp?](http://www.addison-wesley.de/main/main.asp?page=deutsch/bookdetails&ProductID=131885&piacode=&pszielgruppe=)

[page=deutsch/bookdetails&ProductID=131885&piacode=&pszielgruppe=](http://www.addison-wesley.de/main/main.asp?page=deutsch/bookdetails&ProductID=131885&piacode=&pszielgruppe=)

Addison-Wesley

ISBN 978-3-8273-2615-7

erschienen 12.2007

ca. 360 Seiten

Mit CD

Inhaltsverzeichnis

Umstiegsanleitung für JavaFX Script auf die Finalversion 1.0	1
Operationen.....	3
Attributinitialisierung.....	4
Ersetzen von Triggern.....	4
Kardinalität.....	5
Benannte Instanzen	5
Anonyme Objekliterale	6
Überschreiben von Funktionen nun mit expliziter Angabe des Rückgabetyps und override.....	7
for-Schleife	7
Negative Rangebereiche und Schrittweite.....	8
Stringliterale und Stringausdrücke in JavaFX	8
Auswertung von Ausdrücken in Strings.....	9
Zeilenumbrüche in Strings.....	9
Modulooperator.....	9
Arrays.....	9
Einfügen von Arrayelementen.....	10
Die Grundlagen einer GUI mit JavaFX Script.....	10
Bidirektionales Binden - with inverse.....	11
Casting von Number nach Integer	11
Vererbung	11
Das neue API.....	12

Wer bereits mit JavaFX Script auf Basis erster Preversionen seit etwa Mitte 2007 programmiert hat, wird in der nun Ende 2008 erschienenen **Finalversion 1.0** mit einigen Änderungen bzw. Umstrukturierungen konfrontiert.

Etwas bedauerlich ist, dass die Umstrukturierungen über die zweite Hälfte des Jahres 2008 in mehreren Schritten vollzogen wurde und immer wieder Syntaxstrukturen verändert wurden. Insbesondere wurden auch viele Token und Syntaxstrukturen, die in ersten Versionen verwendet wurden, in der Finalversion **aufgegeben**. Insgesamt ist m.E. die Finalversion von JavaFX Script aber viel stringenter und logischer als die Preversionen geworden und orientiert sich doch verstärkter an dem streng objektorientierten Konzept von Java selbst. Allerdings sind auch durchaus syntaktische Anleihen aus anderen Sprachen wie C/C++/C# oder PHP und JavaScript erkennbar.

Operationen

In den Preversionen von JavaFX Script hat Sun mit **Prozeduren** experimentiert. Schon Funktionen sind in der streng objektorientierten Welt von Java gewagt, aber Prozeduren, die sogar einen **Rückgabewert** liefern konnten, waren wohl doch des Guten zu viel. In der Finalversion entfallen diese Prozeduren, die bisher mit dem Schlüsselwort *operation* eingeleitet wurden.

Ein Umstieg von altem Code ist jedoch denkbar einfach. Stattdessen werden einfach grundsätzlich **Funktionen** verwendet, die mit dem Schlüsselwort *function* eingeleitet werden. Man muss dabei also nur konsequent den Token *operation* durch den Token *function* ersetzen. Das ist alles.

Ein Beispiel für die alte Syntax:

```
class MeineKlasse {  
    function times2(x) { return x * 2; }  
    operation print(s) { System.out.println(s); }  
}
```

Das gleiche Beispiel in der neuen Syntax:

```
class MeineKlasse {  
    function times2(x) { return x * 2; }  
    function print(s) { System.out.println(s); }  
}
```

Eine Funktion kann auch explizit mit *Void* gekennzeichnet werden um zu kennzeichnen, dass eine Funktion keinen Rückgabewert liefert.

Beispiel:

```
function ausgabe() : Void {  
    println("Keine Rückgabe!");  
}
```

Attributinitialisierung

Die **Initialisierung** von **Attributen** erfolgte bisher außerhalb einer Klassendefinition. Diese Technik wurde nun Java angeglichen und die Attribute werden bei der Deklaration in der Klasse selbst initialisiert.

Ein Beispiel für die alte Syntax:

```
class MeineKlasse {  
    attribute bar: Boolean; }  
  
attribute Foo.bar = true;
```

Das gleiche Beispiel in der neuen Syntax:

```
class MeineKlasse {  
    var bar: Boolean = true;  
}
```

Beachten Sie, dass nun auch der Token *attribute* entfällt und stattdessen *var* (oder *def* beim Einsatz von dynamischem Binden) zum Einsatz kommt.

Ersetzen von Triggern

Bisher waren **Trigger**, die außerhalb von einer Klasse definiert wurden, ein zentraler Bestandteil der Definition von der Reaktion auf **Ereignisse**. Die neue Syntax definiert so genannte **replace triggers** als Teil der Attributdeklaration. Die Trigger-Funktion folgt dabei nun den Schlüsselworten *on* *replace*.

Ein Beispiel für die alte Syntax:

```
class MeineKlasse {  
    attribute bar: Boolean;  
}  
  
trigger on MeineKlasse.bar = value {  
    if (bar == true) {  
        beep();  
    }  
}
```

Das gleiche Beispiel in der neuen Syntax:

```
class MeineKlasse {  
    var bar: Boolean on replace {  
        if (bar == true) { beep(); }  
    };  
}
```

Ein Beispiel für die alte Syntax mit Initialisierung:

```

class MeineKlasse {
  attribute bar: Boolean = true;
}

trigger on MeineKlasse.bar = value {
  if (bar == true) {
    beep();
  }
}

```

Das gleiche Beispiel in der neuen Syntax mit Initialisierung:

```

class MeineKlasse {
  var bar: Boolean = true on replace {
    if (bar == true) {
      beep();
    }
  };
}

```

Kardinalität

Die beliebige **Häufigkeit** von Attributen wurde bisher mit dem Sternoperator * angegeben und die Werte außerhalb zugewiesen. Nun werden dafür die eckigen Klammern [] verwendet und die Werte direkt in der Klasse zugewiesen.

Ein Beispiel für die alte Syntax:

```

class MeineKlasse {
  attribute names :String*;
}

attribute names = ["Ralph", "Felix", "Florian"];

```

Das gleiche Beispiel in der neuen Syntax:

```

class MeineKlasse {
  var names :String[] = ["Ralph", "Felix", "Florian"];
}

```

Benannte Instanzen

In den bisherigen Versionen von JavaFX Script waren **benannten Instanzen** mehr oder weniger das, was nun vollqualifizierte Konstanten darstellen (oder **Objektliterale**).

Ein Beispiel für die alte Syntax:

```
Frame {
  title: "White Frame"
  background: white
}
```

Ein vergleichbares Beispiel in der neuen Syntax:

```
SwingLabel {
  text: "Label"
  foreground: Color.WHITE
}
```

Alternativ das gleiche Beispiel in der neuen Syntax:

```
SwingLabel {
  text: "Label"
  foreground: Color {
    red: 1
    green: 1
    blue: 1
    opacity: 1
  }
}
```

Anonyme Objeklitterale

Bisher konnten Sie ein **anonymes Objeklitteral** ohne die Spezifizierung von einem Typ deklarieren. Der Interpreter legte den Typ implizit fest (eine Art der losen Typisierung, wie es auch unter JavaScript oder PHP Einsatz findet, unter Java allerdings ziemlich brutal im Widerspruch zum streng typisierten Konzept stand). In der neuen Version müssen Sie explizit Objeklitterale benennen.

Ein Beispiel für die alte Syntax:

```
accelerator: {
  modifier: CTRL
  keyStroke: O
}
....
```

Das gleiche Beispiel in der neuen Syntax:

```
accelerator: Accelerator {
  modifier: KeyModifier.CTRL
  keyStroke: KeyStroke.O
}
...
```

Überschreiben von Funktionen nun mit expliziter Angabe des Rückgabetyps und override

Bisher konnten Sie beim Überschreiben von einer Funktion diese einfach redefinieren und zum Beispiel auf die Angabe des **Rückgabetyps** verzichten.

Nun geben Sie in der Signatur der Funktion diesen Rückgabotyp und das vorangestellte Schlüsselwort *override* an. Zwar ist so eine vollständige Signatur nicht in jedem Fall unbedingt zwingend, aber zur Kennzeichnung und Vereinheitlichung immer sinnvoll.

Ein Beispiel für die alte Syntax:

```
class MyWidget extends CompositeNode {  
    ...  
    function composeNode() {  
        ...  
    }  
}
```

Ein vergleichbares Beispiel in der neuen Syntax:

```
class MyCustomNode extends CustomNode {  
    public override function create(): Node {  
        return Group {  
            content: []  
        };  
    }  
}
```

for-Schleife

Nachdem man in Java selbst ein **foreach**-Konstrukt eingeführt, aber syntaktisch anders als in JavaFX Script aufgebaut hat, war diese syntaktische Abweichung in JavaFX Script ziemlich unglücklich. In der neuen Varianten wurde die Syntax an Java angepasst.

Ein Beispiel für die alte Syntax:

```
...  
for (Integer i = 0; i < element.length; i++) {  
    System.out.println(element);  
}  
...  
foreach (element in group) {  
    System.out.println(element);  
}  
...
```

Das gleiche Beispiel in der neuen Syntax:

```
...
for (Integer i = 0; i < element.length; i++) {
    System.out.println({element});
}
...
for (element in group where element.length() < 4) {
    System.out.println({element});
}
...
```

Negative Rangesbereiche und Schrittweite

Bisher ging so was:

```
for (i in [5..4]) {
    System.out.println("i = {i}");
}
```

In der neuen Version sind **negative Rangesbereiche** nicht mehr erlaubt.

Ebenso können Sie die **Schrittweite** der Iteration nicht mehr wie folgt angeben:

```
for (i in [1,5..10]) {
    System.out.println("i = {i}");
}
```

Stattdessen verwendet man nun den Token *step*, der so auch normal in Arrays zu verwenden ist.

Beispiel:

```
var nums = [1..10 step 2];
```

Stringlitterale und Stringausdrücke in JavaFX

Sie können in JavaFX Script eine Zeichenkette sowohl in **einfache** als auch **doppelte** Hochkommata einschließen. Und wie auch beispielsweise in PHP wurden Zeichenketten in den ersten Versionen von JavaFX Script in doppelten und einfachen Hochkommata vom System bei gewissen Situationen unterschiedlich behandelt.

Auswertung von Ausdrücken in Strings

Allgemein gilt - wenn Sie den Bezeichner einer Variablen in eine Zeichenkette mit doppelten Hochkommata notieren und diesen in geschweifte Klammern `{}` einschließen, wird der Ausdruck **ausgewertet**. Sie erhalten in dem resultierenden String den Wert der eingeschlossenen Variablen beziehungsweise eines Ausdrucks. Dabei sind Sie für die Art der ausgewerteten Variable nicht auf Zeichenketten beschränkt. Bei einer Zeichenkette in einfachen Hochkommata finden Sie in bisherigen Versionen dieses Verhalten **nicht**.

Beispiel:

```
import java.lang.System;
var name = 'Trillian';
var wert = 5;
var s1 = "Hallo {name}"; // Zeichenkette mit doppelten Hochkommata
var s2 = 'Hallo {name}'; // Zeichenkette mit einfachen Hochkommata
var s3 = "Hallo {wert}"; // Zeichenkette mit doppelten Hochkommata

System.out.println(s1); // Hallo Trillian
System.out.println(s2); // Hallo {name}
System.out.println(s3); // Hallo 5
```

Die Ausgabe ist in **alten Versionen** folgende:

```
Hallo Trillian
Hallo {name}
Hallo 5
```

In **neuen Versionen** erhalten Sie aber folgende Ausgabe:

```
Hallo Trillian
Hallo Trillian
Hallo 5
```

M.a.W.: Auch in einfachen Hochkommata eingeschlossene Ausdrücke werden in der neuen Version von JavaFX Script ausgewertet!

Zeilenumbrüche in Strings

Im Gegensatz zu Java konnten in JavaFX in den ersten Versionen Zeichenketten Zeilenumbrüche enthalten, wenn sie in doppelte Hochkommata eingeschlossen sind. Das geht nun nicht mehr.

Modulooperator

Der Modulooperator `%` wurde in neuen JavaFX-Versionen durch den Token `mod` ersetzt. Damit ist allerdings eine weitere Inkompatibilität zu Java zu beachten.

Arrays

In JavaFX ist `[]` ein eigener Operator und dieser Operator `[]` steht für eine **Selektion**. Die darin

notierte Lokalisierung ist eine Verwendung, die ähnlich in XPath beziehungsweise XQuery vorkommt.

Einfügen von Arrayelementen

Das Einfügen von Elementen erfolgt über die *insert*-Anweisung. Diese Anweisung konnte bisher die folgenden verschiedene Formen annehmen:

- *insert Ausdruck1 [as first | as last] into Ausdruck2*
- *insert Ausdruck1 before Ausdruck2*
- *insert Ausdruck1 after Ausdruck2*

Die Token *as first* und *as last* wurden aufgegeben!

Die Grundlagen einer GUI mit JavaFX Script

Die grundlegenden Klassen zum Aufbau einer GUI wurden in der Finalversion von JavaFX Script massiv unstrukturiert, ohne jedoch das grundsätzliche Konzept zu verändern. Die Basis bilden nur neue Klassen.

Unter *javafx.ext.swing* sind nun die meisten Swing-Komponenten zu finden.

Und die Grundlage der gesamten grafischen Applikation bildet *javafx.stage.Stage*, was für das gesamte Fenster steht.

Dieses wird in der Regel in einzelne Szenen (*javafx.scene.Scene*) mit spezifischen Inhalten (also etwa Swing-Komponenten) unterteilt werden.

Beispiel:

```
import javafx.ext.swing.SwingButton;
import javafx.scene.Scene;
import javafx.stage.Stage;
```

```
var button = SwingButton {
    text: "Button"
    action: function() {
        java.lang.System.exit(0);
    }
}
```

```
}
Stage {
    title: "Meine Applikation"
    width: 250
    height: 80
    scene: Scene {
        content: button
    }
}
```

Bidirektionales Binden - with inverse

Das **bidirektionales Binden** verwendet in der neuen Variante von JavaFX die Syntax *with inverse*.

Ein Beispiel für die alte Syntax:

```
... TextField {  
  value: bind model.firstName  
}  
...
```

Ein vergleichbares Beispiel in der neuen Syntax:

```
...  
SwingTextField {  
  columns: 10  
  text: bind model.firstName with inverse  
  editable: true  
}  
...
```

Casting von Number nach Integer

Bei der Typumwandlung von *Number* nach *Integer* verwendet man in neuen JavaFX-Varianten die Funktion *intValue()*, um einem Verlust der Genauigkeit vorzubeugen.

Ein Beispiel für die alte Syntax:

```
var real :Number;  
num = 6.42;  
var integer :Integer;  
integer = real;  
...
```

Das gleiche Beispiel in der neuen Syntax:

```
var real :Number;  
var num = 6.42;  
var integer :Integer = real.intValue();  
...
```

Vererbung

Bei der **Vererbung** gibt es in JavaFX Script aktuell ein paar kleinere Probleme, so dass das Schlüsselwort *as* in einigen Situationen notwendig sein kann.

Beispiel:

...

content: foo as Node

...

Das neue API

Das **API** von JavaFX Script wurde wie schon erwähnt in einigen Bereichen reorganisiert. Insbesondere zur GUI. Details lassen sich aber leicht über die JavaFX Script Programming Language API Documentation unter <http://java.sun.com/javafx/1/docs/api/> nachschlagen.

Stichwortverzeichnis

Anonyme Objekliterale	6
Arrays.....	9
as first.....	10
as last.....	10
attribute.....	4
Attributinitialisierung.....	4
Benannte Instanzen.....	5
Bidirektionales Binden	11
Casting.....	11
def.....	4
for-Schleife	7
foreach.....	7
function.....	3
Funktionen.....	3
GUI.....	10
insert.....	10
intValue().....	11
inverse.....	11
javafx.ext.swing.....	10
javafx.scene.Scene.....	10
javafx.stage.Stage.....	10
Kardinalität.....	5
mod.....	9
Modulooperator.....	9
on replace.....	4
operation.....	3
Operationen.....	3
override.....	7
Prozeduren.....	3
Rangebereiche.....	8
replace triggers.....	4
Scene.....	10
Schrittweite.....	8
Selektion.....	9
Stage.....	10
step.....	8
Stringausdrücke.....	8
Stringliterale.....	8
Swing.....	10
Szenen.....	10
Trigger.....	4
Überschreiben von Funktionen.....	7
var.....	4
Vererbung.....	11
Void.....	3

with inverse.....	11
Zeilenumbrüche in Strings.....	9
[].....	5, 9
*.....	5
%.....	9