JavaFX

Wolfgang Weigend
ORACLE Deutschland B.V. & Co. KG

Entwicklung mit JavaFX

Für die Entwicklung von Client-Web-Anwendungen wurde JavaFX von Oracle als strategische Technologie innerhalb der Java Plattform für den Desktop vorgestellt. JavaFX 2 erfährt eine Renaissance mit nachhaltiger Verstärkung der zuständigen Entwicklungs-Teams und klar definierter Roadmap für die kommenden Versionen mit dem Ziel, JavaFX im Java SE Development Kit (JDK) 8 zu paketieren und auszuliefern. Die aktuelle Version JavaFX 2.2 wird seit dem JDK 7u6 und höheren JDK-Versionen gebündelt für MS Windows, Mac OS X und Linux ausgeliefert.

Die in JavaFX 1.0 geschaffene propritäre Skriptsprache JavaFX Script ist mit JavaFX 2.0 vollständig verschwunden und man setzt jetzt auf eine einheitliche Java-Entwicklung mit neuer JavaFX Bibliothek für den Desktop. Das Augenmerk liegt dabei auf verbesserter Browser-Integration, dem UI-Styling, Animationen und höherer Geschwindigkeit der Grafikverarbeitung. Die verfügbaren API´s sind vereinfacht worden, sodass sie den Java-Entwicklungsprozess mit seinen Werkzeugen besser unterstützen. Damit gestaltet sich der Einsatz von JavaFX deskriptiv und programmatisch und führt zu einer schnelleren Umsetzung bei der Entwicklung von Desktop-Anwendungen. JavaFX 2.0 liefert einige UI-Controls (Charts, Tabellen, Menüs) und ein API für selbst erstellte Controls, mit der Absicht weitere folgen zu lassen, um den konkreten Bedarf an ausgereifter Desktop-Funktionalität zu decken und langfristig Swing abzulösen.

Java als Programmiermodell für JavaFX 2

Die Einführung von Java API's für JavaFX bietet mit Generics, Annotationen, und Multithreading die gewohnte Java-Programmiermodellunterstützung mit vorhandenen Sprachmerkmalen an. Beim Design der API's versuchte man bewußt eine Alternative zu dynamisch typisierten Sprachen zu schaffen. Die Funktionalität von JavaFX wird dem Entwickler über die API's direkt angeboten, um vorhandene Java-Werkzeuge für Code Refactoring, Debugging und Profiling, auch für die Erstellung von JavaFX-Anwendungen nutzen zu können.

Architektur für schnelle Grafikverarbeitung

Die neue JavaFX Grafik-Engine unterstützt moderne Grafikprozessoren (Graphics Processing Units). Die Hardware-beschleunigte Grafik-Pipeline Prism bildet das Fundament der Grafik-Engine für schnelles Rendering von 2D- und 3D-Elementen (siehe Abbildung 1). Web-Seiten die in einer JavaFX Anwendung als Web-Komponente eingebunden sind, werden mittels Web-Kit HTML Rendering Technologie und Prism dargestellt. Prism spielt mit dem Glass Windowing-Toolkit zusammen. Damit macht es die Darstellung umfangreicher grafischer Oberflächen schneller und wird in getrennten Threads für die Anwendung und das Rendering im Hintergrund synchronisiert, ohne das es der Entwickler bemerkt. Ein JavaFX Plug-In ermöglicht das Laden von Prism-basierten Applets im

Browser. Die neue JavaFX Media Engine unterstützt den schnellen Ablauf von Web Multimedia Content Video und Audio in der Playback-Funktion mit dem GStreamer Multimedia Framework.

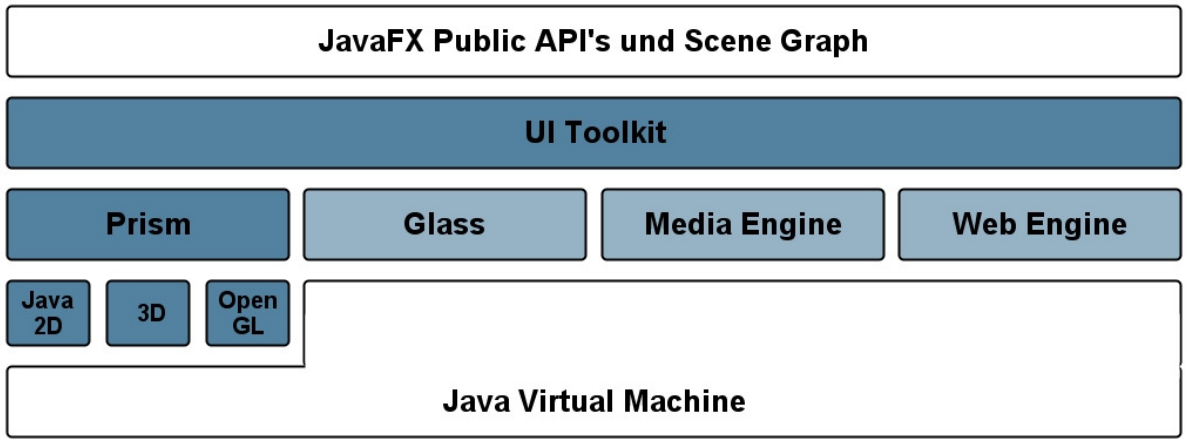

Abbildung 1: JavaFX Architektur

Entwicklungsumgebungen und Scene Builder

Mit nahezu jeder Java Entwicklungsumgebung kann man JavaFX verwenden. Zur Auswahl stehen NetBeans, Eclipse IDE über das JavaFX-Eclipse-Integrationsprojekt e(fx)clipse, IntelliJ IDEA und JDeveloper/ADF. JavaFX-Unterstützung existiert auch für Groovy, über das GroovyFX API und für Scala mit ScalaFX (Scala Bindings for JavaFX 2). Die Verwendung von JavaFX in der Entwicklungsumgebung NetBeans mit MS Windows XP Betriebssystem, erfolgt durch Konfiguration mit dem Java Plattform Manager über die Definition einer Java Plattform Version (JDK 6 oder JDK 7) und Einbindung der *JavaFX 2.0 Runtime* mit der Datei `fxrt.jar` aus dem Verzeichnis `C:\Program Files\Java\jdk1.7.0_06\jre` (siehe Abbildung 2).

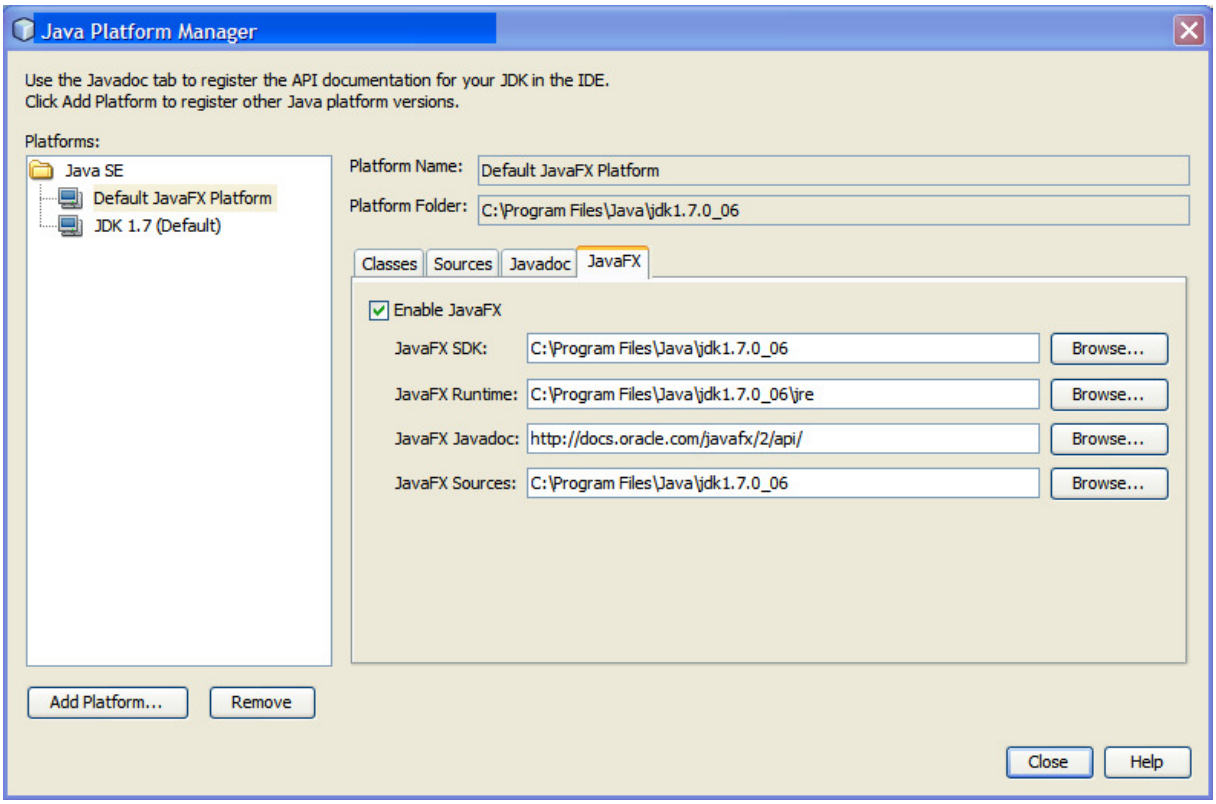

Abbildung 2: NetBeans Java Plattform Manager mit JavaFX

Durch Einführung der neuen XML-basierten deklarativen Markup Sprache FXML, kann das Benutzer-Interface der JavaFX-Anwendung unabhängig von der jeweils verwendeten Entwicklungsumgebung aufgebaut werden, ohne IDE-spezifische Fragmente nachpflegen zu müssen. Bei Layout-Änderungen ist das vorteilhaft, da keine erneute Code-Kompilierung notwendig ist.

Der neue JavaFX Scene Builder 1.0 (siehe Abbildung 3), ist ein visuelles Layout-Werkzeug zum Design von JavaFX Anwendungsoberflächen ohne kodieren zu müssen. Im Scene Builder Editor können UI-Komponenten per drag und drop im Arbeitsfeld der Oberfläche platziert werden, deren Eigenschaften verändert und Style-Sheets zugeordnet werden können. Für das gewählte Layout wird automatisch das passende FXML erzeugt und in einer FXML-Datei abgelegt. Sie kann dann in ein Java-Projekt einer Entwicklungsumgebung integriert werden. Die JavaFX-Oberfläche kann dort mit der Applikationslogik verbunden werden. Mit dem eingebauten Scene Builder Preview kann im Menü über *Preview in Window* die Oberfäche von `IssueTracking.fxml` auf dem Desktop sichtbar gemacht und gemäß ihrer Funktionalität verwendet werden.

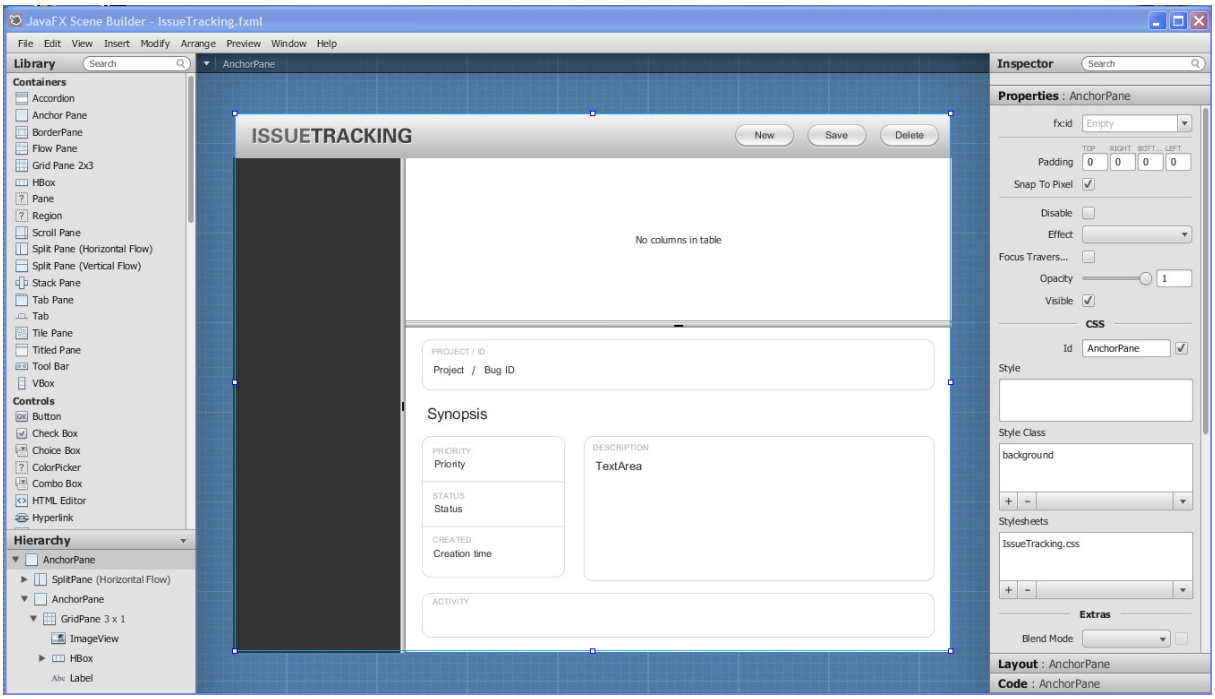

Abbildung 3: JavaFX Scene Builder 1.0

Wie im JavaFX Scene Graph in Abbildung 4 dargestellt, verwendet JavaFX zuerst eine Stage, um die Bühne zu eröffnen und darin eine Scene mit Parent- und Child-Nodes anzuordnen. Die Scene wird mit Layout und Controls verknüpft und in der JavaFX-Anwendung über ein Event-Framework gesteuert. Dies zeigt das Code-Beispiel *javafxapplication1* (Hello World) im Listing 1, bei der ein Button „`Say Hello World`“ in einer Oberfläche erscheint und beim Betätigen der gedrückte Zustand über ein *ActionEvent* vom Event Handler ausgewertet wird und im IDE-Output den String „`Hello Alexander!`“ ausgibt. Die JavaFX Klasse `Scene (public class javafx.scene.Scene)` ist der Ursprung von jeglichem Content im Scene Graph. Der Scene-Hintergrund wird gefüllt indem die Variable `fill` angegeben wird. Die Gruppe der Nodes einer Content-Sequenz wird dann für die Scene gerendert.

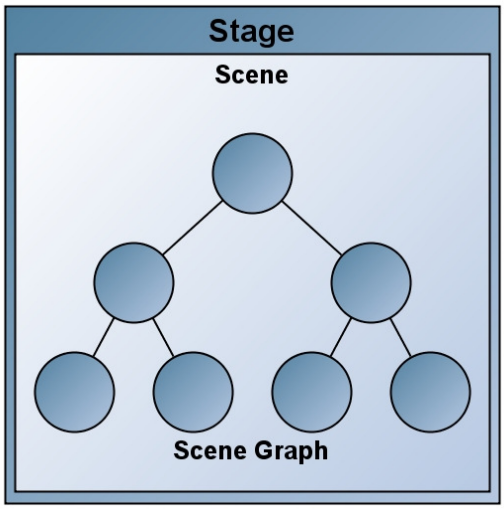

Abbildung 4: JavaFX Scene Graph

Listing 1

```
package javafxapplication1;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class JavaFXApplication1 extends Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello Alexander!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

Mit der Beispielanwendung JavaFX Ensemble, wird eine Auswahl von Anwendungen für den Entwickler bereitgestellt, um einzelne JavaFX Merkmale wie Animationen, Charts, Controls, Grafiken in 2D & 3D, Layout, Media, HTML, Web View zu demonstrieren und den Quell-Code in die Entwicklungsumgebung übernehmen zu können. Das Code-Beispiel *Sequential Transition* (Listing 2) zeigt anschaulich, wie sequenzielle Übergänge in JavaFX programmiert werden. Ein grüner Würfel (Abbildung 5) wird dabei ein- und ausgeblendet, jeweils nach rechts und links waagerecht verschoben, um 180° nach rechts und links gedreht, vergrößert und verkleinert, wieder nach rechts und links gedreht, nach rechts und links verschoben, usw. Zuerst wird der Würfel als *Rectangle* erzeugt und dann werden die vier Übergänge `fadeTransition` (Ein-/Ausblenden), `translateTransition` (Seitwärtsbewegung), `rotateTransition` (Drehung), `scaleTransition` (Verkleinern/Vergrößern) nacheinander durchlaufen.

Listing 2

```
/* JavaFX Ensemble Sequential Transition */
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.event.EventHandler;

import javafx.animation.*;
import javafx.scene.Node;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.util.Duration;

/**
 * A sample in which various transitions are executed sequentially.
 *
 * @related animation/transitions/ParallelTransition
 * @see javafx.animation.SequentialTransition
 * @see javafx.animation.Transition
 */
public class SequentialTransitionSample extends Application {

    private SequentialTransition sequentialTransition;

    private void init(Stage primaryStage) {
        Group root = new Group();
        primaryStage.setResizable(false);
        primaryStage.setScene(new Scene(root, 400,100));
        // create rectangle
        Rectangle rect = new Rectangle(-25,-25,50, 50);
        rect.setArcHeight(15);
        rect.setArcWidth(15);
        rect.setFill(Color.GREEN);
        rect.setTranslateX(50);
        rect.setTranslateY(50);
        root.getChildren().add(rect);
        // create four transitions as fade, translate, rotate and scale
        FadeTransition fadeTransition =
            new FadeTransition(Duration.seconds(1), rect);
        fadeTransition.setFromValue(1.0f);
        fadeTransition.setToValue(0.3f);
        fadeTransition.setCycleCount(2);
        fadeTransition.setAutoReverse(true);
        TranslateTransition translateTransition =
```

```
            new TranslateTransition(Duration.seconds(2), rect);
        translateTransition.setFromX(50);
        translateTransition.setToX(375);
        translateTransition.setCycleCount(2);
        translateTransition.setAutoReverse(true);
        RotateTransition rotateTransition =
            new RotateTransition(Duration.seconds(2), rect);
        rotateTransition.setByAngle(180f);
        rotateTransition.setCycleCount(4);
        rotateTransition.setAutoReverse(true);
        ScaleTransition scaleTransition =
            new ScaleTransition(Duration.seconds(2), rect);
        scaleTransition.setToX(2);
        scaleTransition.setToY(2);
        scaleTransition.setCycleCount(2);
        scaleTransition.setAutoReverse(true);
        // create sequential transition to do four transitions one after
another
        sequentialTransition = new SequentialTransition();
        sequentialTransition.getChildren().addAll(
                fadeTransition,
                translateTransition,
                rotateTransition,
                scaleTransition);
        sequentialTransition.setCycleCount(Timeline.INDEFINITE);
        sequentialTransition.setAutoReverse(true);
    }

    public void play() {
        sequentialTransition.play();
    }

    @Override public void stop() {
        sequentialTransition.stop();
    }

    public double getSampleWidth() { return 400; }

    public double getSampleHeight() { return 100; }

    @Override public void start(Stage primaryStage) throws Exception {
        init(primaryStage);
        primaryStage.show();
        play();
    }
    public static void main(String[] args) { launch(args); }

}
```

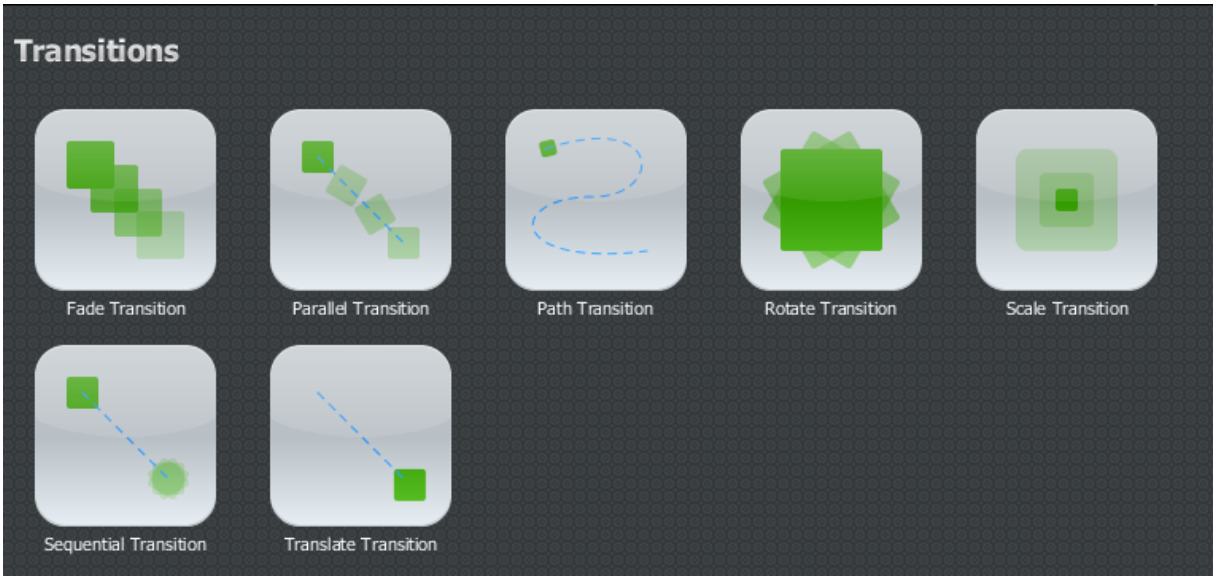

Abbildung 5: JavaFX Ensemble Transitions-Anwendungsbeispiele

Bei früheren Rechnergenerationen mit Problemen bei der 3D-Dastellung kann der Ablauf von 3D Funktionaliät in den Netbeans IDE Projekt Eigenschaften unter der Kategorie *Run* mit den Einstellungen der *VM Options:* `-Dprism.forceGPU=true` explizit gesetzt werden, sodass die Ensemble Anwendung mit den Graphics 3d Beispielen *Cube* und *Cube System* ablauffähig ist.

Die beschriebenen Beispiele sind per Download der Datei `javafx_samples-2_2_0.zip` als *JavaFX 2.2 Samples* für MS Windows, Mac OS X und Linux verfügbar und können mit der Entwicklungsumgebung verwendet werden. Als nützliche Informationsquelle erweist sich *FX Experience* (http://fxexperience.com/) mit technischen Erfahrungsberichten, Verfügbarkeit neuer UI-Controls und weiteren Beispielen.

JavaFX in Swing Anwendungen einbinden

Die Einbindung von JavaFX in Swing Anwendungen erfolgt mit der JFXPanel Komponente (siehe Abbildung 6). Die Klasse JFXPanel (`public class JFXPanel, extends javax.swing.JComponent`) ist im Listing 3 aufgeführt. Der darzustellende Inhalt ist mit der Methode `setScene(javafx.scene.Scene)` spezifiziert, die eine Instanz der JavaFX Scene erhält. Nach dem Zuordnen der Scene wird diese automatisch nachgezeichnet und alle Input- und Fokus-Events gelangen direkt zur Scene. Bei der Verwendung von JFXPanel gibt es die Einschränkung, dass die Swing Komponente nur über den Event-Dispatch-Thread erreichbar sein sollte. Einzige Außnahme ist der Aufruf der Methode `setScene(javafx.scene.Scene)` vom Event-Dispatch-Thread oder dem JavaFX-Anwendungs-Thread. Das typische Pattern zur Verwendung von `JFXPanel` wird im Listing 4 gezeigt.

Listing 3

```
Class JFXPanel

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javafx.embed.swing.JFXPanel
```

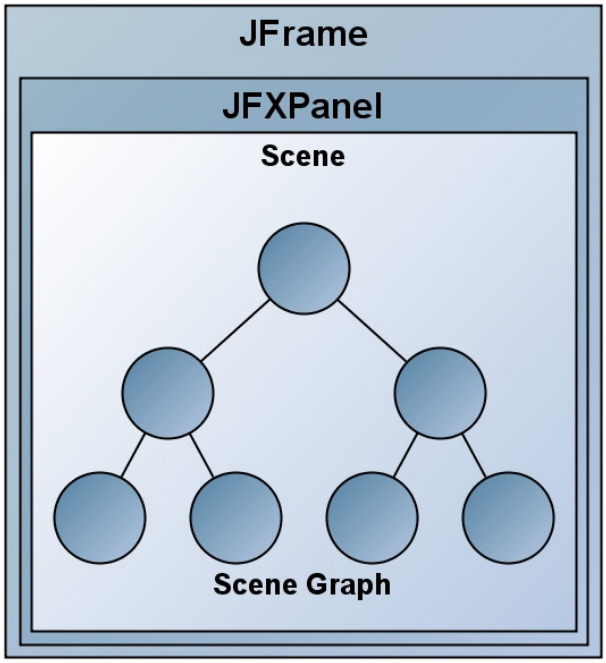

Abbildung 6: JFXPanel Komponente in Swing

Listing 4

```
public class Test {

    private static void initAndShowGUI() {
        // This method is invoked on Swing thread
        JFrame frame = new JFrame("FX");
        final JFXPanel fxPanel = new JFXPanel();
        frame.add(fxPanel);
        frame.setVisible(true);

        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                initFX(fxPanel);
            }
        });
    }

    private static void initFX(JFXPanel fxPanel) {
        // This method is invoked on JavaFX thread
        Scene scene = createScene();
        fxPanel.setScene(scene);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                initAndShowGUI();
            }
        });
    }
}
```

Open Source Projekt OpenJFX

Das Open Source Projekt *OpenJFX* (http://openjdk.java.net/projects/openjfx/), wurde im November 2011 durch die OpenJDK Community etabliert und ist eine wichtige Anlaufstelle für Entwickler, die sich mit JavaFX beschäftigen. Das Ziel vom OpenJFX Projekt ist der Aufbau eines neuen Java Client Toolkits, mit einer eigenständigen Java Spezifikation (JSR), die den JavaFX-Source-Code enthält. Die Beteiligung der Entwicklergemeinschaft ist von erheblicher Bedeutung für JavaFX, um die Einsatzfähigkeit zu erhöhen und die Funktionalität mit weiteren UI-Controls anzureichern.

Fazit

Durch die neu geschaffene technische Ausrichtung von JavaFX und der steigenden Verbreitung in der Java-Community, deutet sich ein positiver Trend für den zu erwartenden Einsatz von JavaFX in Desktop-Anwendungen an. Auch die Verstärkung der Oracle Entwicklergruppe bekräftigt die Substanz von JavaFX und sichert die technische Umsetzung der geplanten vollständigen Distribution von JavaFX 8 (Vormals JavaFX 3.0) mit dem JDK 8. Jedoch entscheidet die Erfahrung der Entwickler im Umgang mit JavaFX und der Reifegrad von JavaFX bis zum Erscheinungszeitpunkt vom JDK 8, über die künftige Verwendung dieser neuen Technologie für Desktop-Anwendungen und wird auch wegweisend für die Ablöse von existierender Swing-Technologie in Java-Anwendungen sein.

Kontaktadresse:

Wolfgang Weigend

ORACLE Deutschland B.V. & Co. KG

Robert-Bosch-Strasse 5

63303 Dreieich

Telefon: **+49 (0) 6106-397-785**

Fax: **+49 (0) 6106-397-105**

E-Mail: **wolfgang.weigend@oracle.com**

Internet: www.oracle.com