kofler.info

JAVA

JAVAFX-PROGRAMM MIT GRADLE

6. MÄRZ 2020 | MICHAEL KOFLER

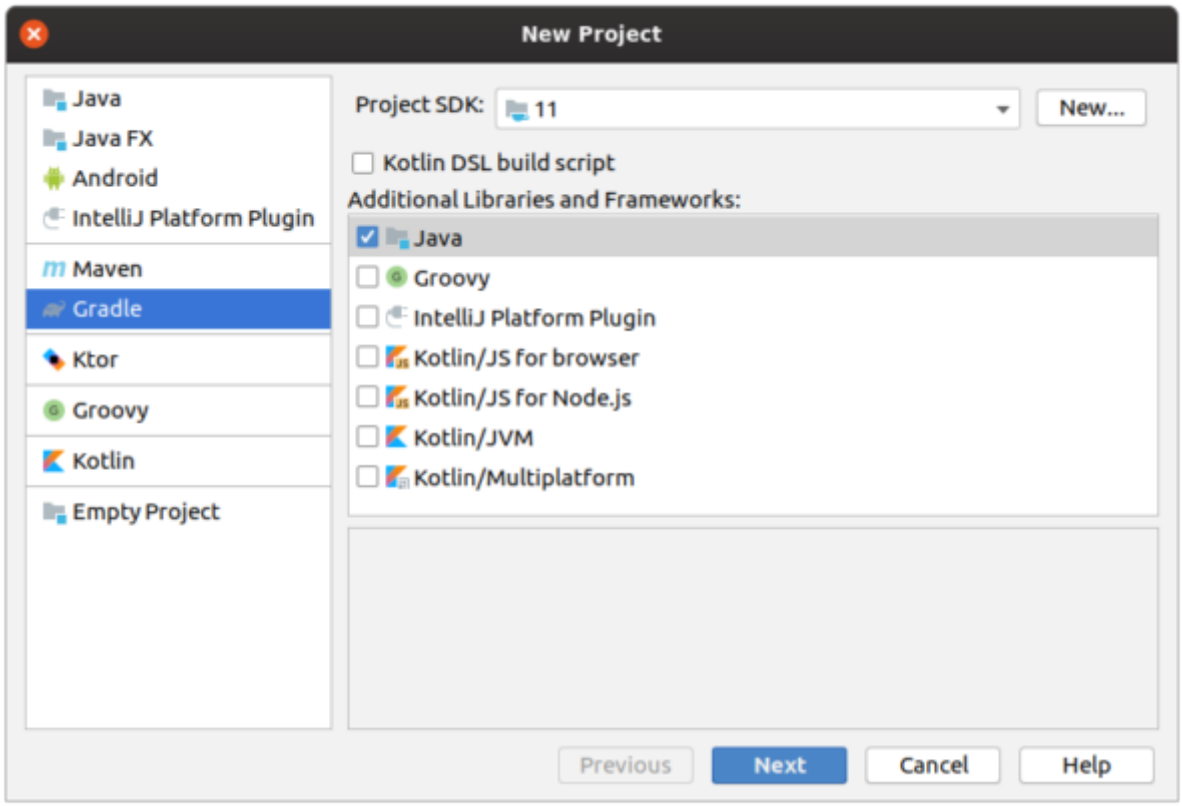
Seit JavaFX aus dem JDK entfernt wurde, ist es nicht mehr ganz einfach, ein minimalistisches JavaFX-Programm zu entwickeln. In der 3. Auflage meines Java-Grundkurses zeige ich, wie Sie die JavaFX-Bibliotheken manuell herunterladen und dann in ein IntelliJ-Projekt integrieren (Veränderung der Modulabhängigkeiten und der VM-Optionen). Das funktioniert, die resultierenden IntelliJ-Projekte laufen wegen der starr eingestellten Pfade aber nicht auf einem anderen Rechner. Besser ist es, das Build-Tool Gradle zu Hilfe zu nehmen.

Im Folgenden setze ich voraus, dass Sie auf Ihrem Rechner das JDK 11 sowie eine aktuelle Version der IntelliJ IDEA installiert haben.

IntelliJ-Projekt mit Gradle einrichten

Gradle ist ein Werkzeug, um Java-Programme zu kompilieren, die auf diverse externe Bibliotheken zurückgreifen. Gradle kommt z.B. standardmäßig zum Einsatz, wenn Sie in Android Studio eine Android-App entwickeln. (Die populärste Alternative zu Gradle ist Maven. Darauf gehe ich hier aber nicht ein.)

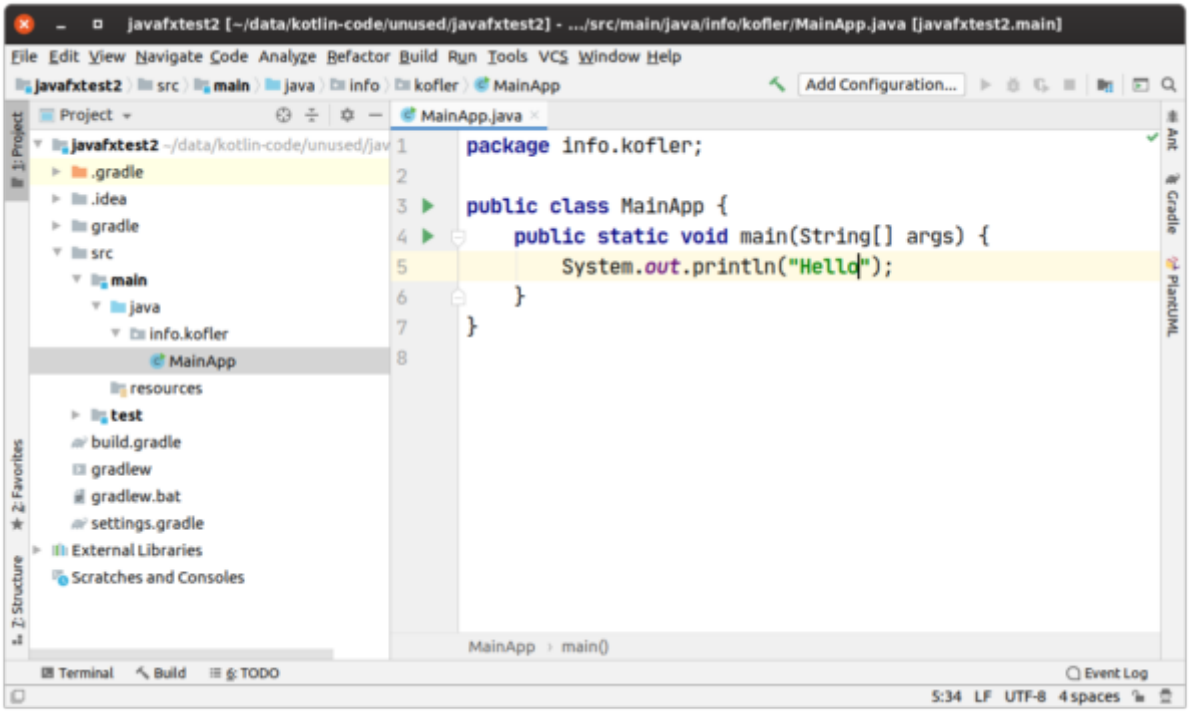
Um in IntelliJ ein neues Projekt zu erstellen, das auf Gradle als Build-Tool zurückgreift, wählen Sie im Projekt *New Project* den Projekttyp *Gradle* und belassen bei *Additional Libraries* die vorselektierte Option *Java*.



In IntelliJ ein Gradle-Projekt einrichten

MainApp.java einrichten

Im nächsten Schritt fügen Sie in der Projektansicht im Pfad `src/main/java` (falls erwünscht) ein neues Package und (auf jeden Fall) Ihre erste Klasse ein. In diesem Beispiel habe ich mich für den Package-Namen `info.kofler` und die Klasse `MainApp` entschieden. Im Editor fügen Sie in `MainApp` die `main`-Methode mit `System.out.println("Hello World")` ein, damit Sie erste Tests (noch ohne JavaFX-Code) durchführen können.

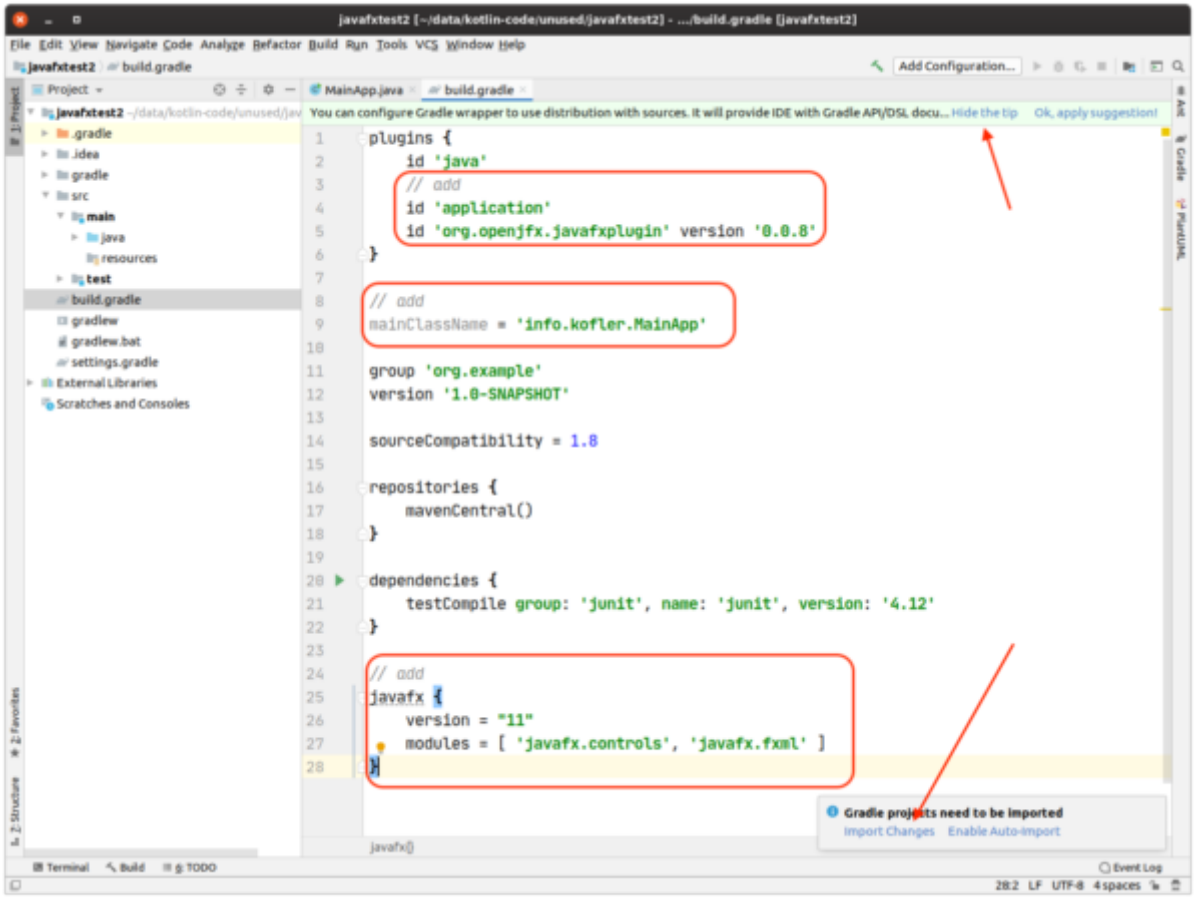


Die erste Version der Klasse »MainApp«

build.gradle ergänzen

Nun öffnen Sie in der Projektansicht die schon vorhandene Datei `build.gradle`.

- Dort müssen Sie die Plugins `application` und `javafxplugin` hinzufügen.
- Sie müssen den Namen Ihrer Start-Klasse angeben. In diesem Beispiel ist das `info.kofler.MainApp`. Geben Sie den Namen Ihrer Klasse angeben, die die `main`-Methode enthält. Die Package-Bezeichnung entfällt, wenn Sie in Ihrem Projekt keine Packages verwenden.
- Schließlich müssen Sie die Gruppe `javafx` hinzufügen, die angibt, welche Version und welche Module der JavaFX-Bibliothek verwendet werden sollen.



build.gradle adaptieren

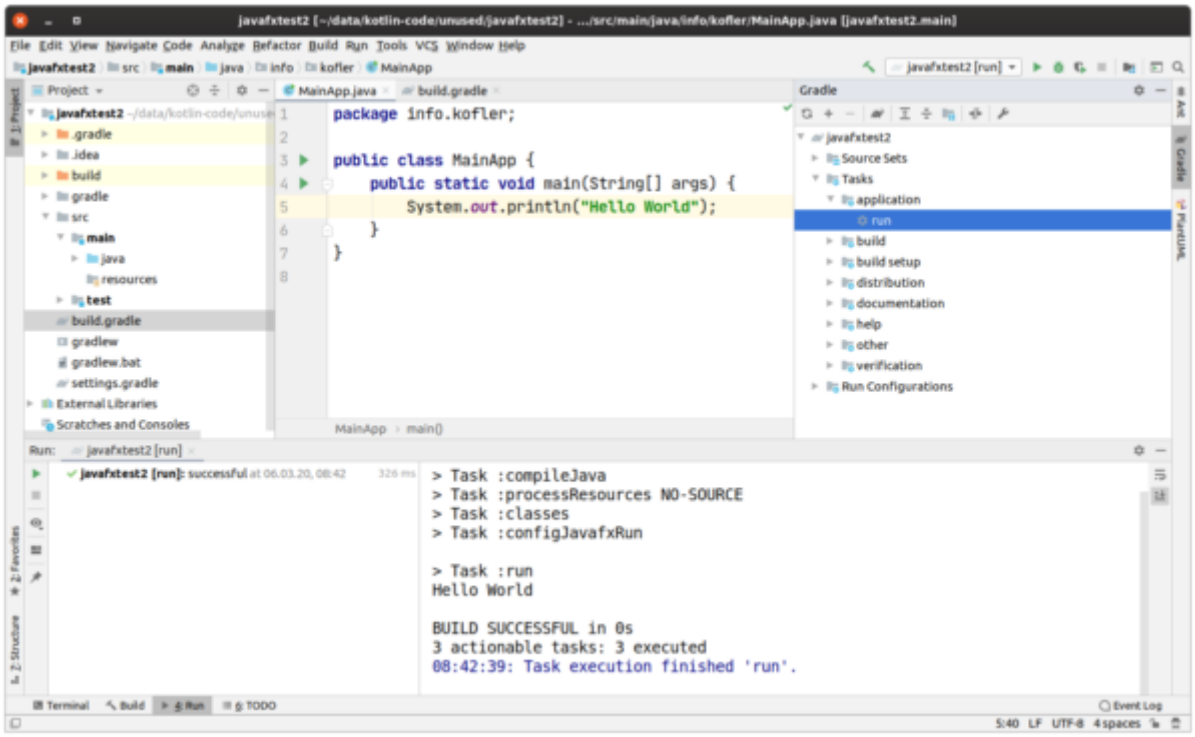
Zuletzt klicken Sie auf *Hide the tip* (Sie wollen den Quellcode von JavaFX *nicht* in Ihr Projekt einbauen) sowie *Import Changes* (d.h., IntelliJ soll die in `build.gradle` durchgeführten Änderungen anwenden).

```
// Mustercode für build.gradle
plugins {
    id 'java'
    // add
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.0.8'
```

```
}

// add
mainClassName = 'info.kofler.MainApp'

group 'org.example'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

// add
javafx {
    version = "11"
    modules = [ 'javafx.controls', 'javafx.fxml' ]
}
```

Erster Test

Um zu testen, ob bis hierher alles funktioniert hat, öffnen Sie in IntelliJ mit *View / Tool Windows / Gradle* das Gradle-Fenster, klappen dort *Tasks / application / run* auf und führen Ihr Projekt durch einen Doppelklick auf *run* aus. Im Run-Fenster sollte die Ausgabe `Hello World` erscheinen. Dann wissen Sie, dass der Build-Prozess prinzipiell funktioniert. Gleichzeitig merkt sich IntelliJ den Prozess zur Programmausführung, d.h., Sie können in Zukunft wie bisher einfach auf den grünen *Run*-Button klicken.



Tasks/application/run im Gradle-Fenster initiiert den ersten Test

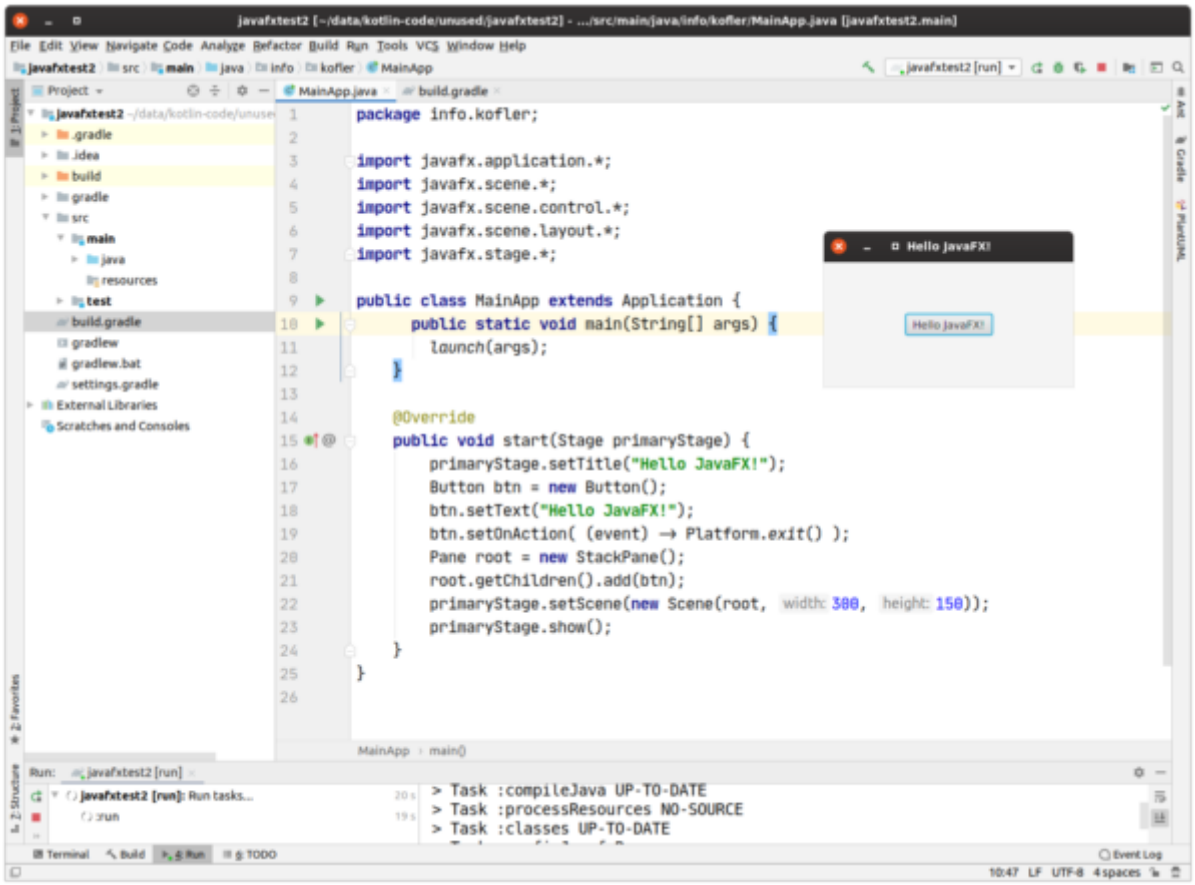
JavaFX-Code

Nun bauen Sie `MainApp.java` dahingehend um, dass anstelle der Hello-World-Textausgabe ein simples JavaFX-Fenster am Bildschirm erscheint. Sie können sich dabei am folgenden Mustercode orientieren. Falls Ihre Datei nicht `MainApp.java` heißt, müssen Sie den Klassennamen entsprechend ändern!

```
// an den eigenen Package-Namen anpassen oder weglassen
package info.kofler;

import javafx.application.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.*;

// MainApp durch den eigenen Dateinamen ersetzen
public class MainApp extends Application {
      public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello JavaFX!");
```

```
        Button btn = new Button();
        btn.setText("Hello JavaFX!");
        btn.setOnAction( (event) -> Platform.exit() );
        Pane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 150));
        primaryStage.show();
    }
}
```

Der *Run*-Button sollte nun Ihr JavaFX-Programm starten.



Hello JavaFX!

Quellen

https://openjfx.io/openjfx-docs/#IDE-Intellij

Download des Musterprojekts

https://kofler.info/wp-content/uploads/javafxtest.zip

GRADLE INTELLIJ JAVA JAVAFX