

Arbeitsblatt 03 JavaFX

1 Einleitung

In diesem Arbeitsblatt lernen Sie die Vorzüge der objektorientierten Programmierung praxisnahe an visuellen Beispielen mithilfe von JavaFX und dem SceneBuilder.

JavaFX ist ein Package (eine Sammlung von Klassen) mit deren Hilfe wir moderne grafische Benutzeroberflächen bzw. moderne grafische Java-Programme erstellen können. Jedes der verwendeten grafischen Elemente z.B. eine Textbox ist eine eigene Klasse und auf ihrem GUI ein eigenständiges Objekt.

Sie können mit dem «.» Operator auf Member des Objekts zugreifen und den Status verändern sowie Operationen durchführen, z.B. `Textbox.select()` (Cursor wird zu einem Strich in der Textbox).

2 JavaFX installieren

Die neusten Versionen der IntelliJ IDEA IDE verwenden im Zusammenhang mit JavaFX Projekten das Projektverwaltungs-Tool Maven. Maven listet die benötigten Abhängigkeiten zu JavaFX bereits auf und übernimmt somit die Integration von JavaFX in unseren Projekten.

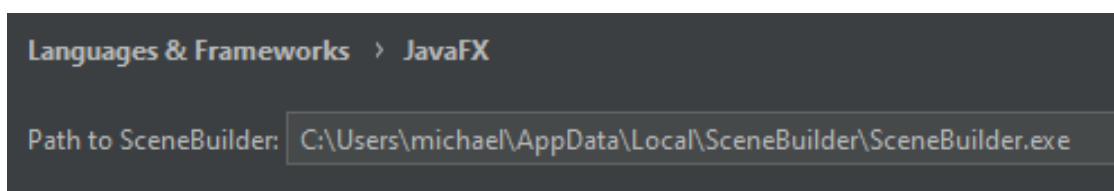
2.1 SceneBuilder installieren

JavaFX beschreibt die grafischen Benutzeroberflächen mithilfe von fxml-Dateien. Diese Files und GUI-Strukturierungen können Sie selbst schreiben, oder Sie verwenden ein hilfreiches Tool, welches Ihnen erlaubt die GUI-Komponenten von JavaFX mithilfe einer grafischen Oberfläche sowie einfacher Drag & Drop Funktionalität zu nutzen.

Die Installations-Source befinden sich im Team: **Modul404 → Kursunterlagen → o7Software → SceneBuilder-18.0.0.msi**.

Nachdem Sie den SceneBuilder installiert haben, müssen diesen im IntelliJ hinterlegen, damit Sie die fxml-Dateien direkt aus der IDE mit dem SceneBuilder öffnen können. Zum Hinterlegen gehen Sie wie folgt vor:

1. Starten Sie IntelliJ
2. Klicken Sie auf Einstellungen
3. Klicken Sie auf Languages & Frameworks
4. Klicken Sie auf JavaFX
5. Hinterlegen Sie den Pfad zur SceneBuilder Applikation



3 Aufgabe: Hello World

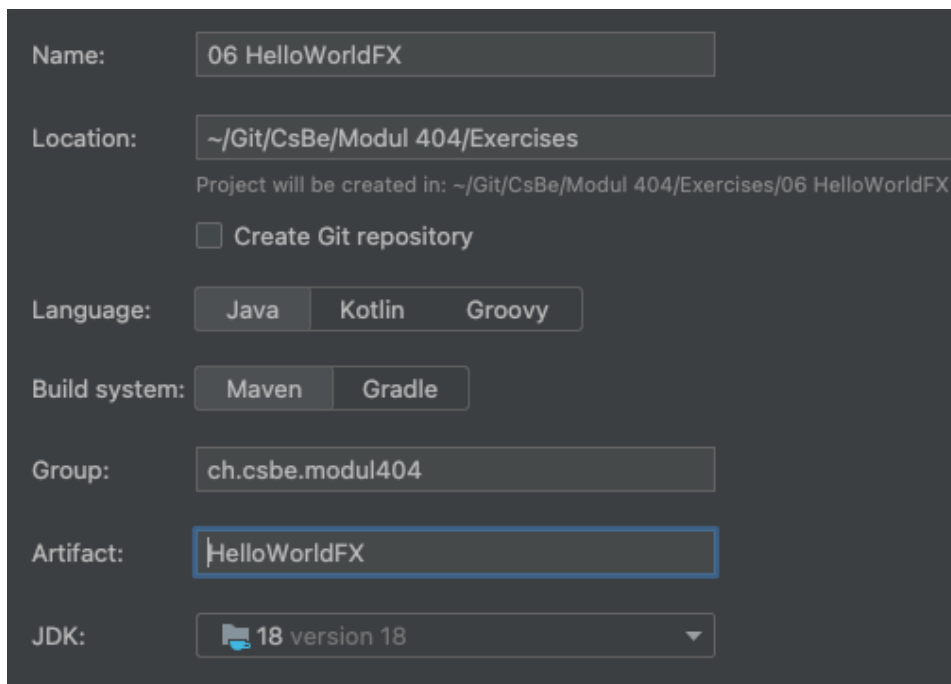
Nun ist es an der Zeit die Grundlegenden Klassen und Objekte von JavaFX kennenzulernen und eine erste grafische Benutzeroberfläche zu kreieren. Hierfür bedienen wir uns der klassischen «Hello World» Aufgabenstellung.

Ziel der Aufgabe:

Erstellen einer grafischen Benutzeroberfläche mit einem Button «Hello Applikation», der wenn er gedrückt wird, verschwindet und einen Text, nämlich «Hallo World» auf dem Fenster anzeigt.

Vorgehen:

1. Erstellen Sie ein neues JavaFX-Projekt in ihrem Repository und benennen Sie es mit **o6HelloWorldFX**



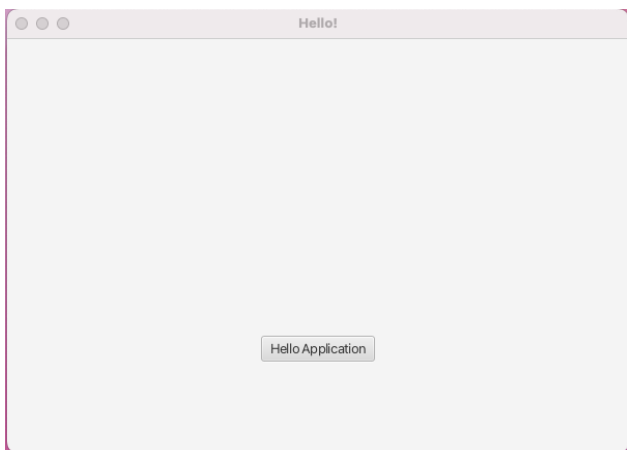
The screenshot shows the 'New Project' dialog in IntelliJ IDEA. The 'Name' field is filled with '06 HelloWorldFX'. The 'Location' field shows the path '~/.Git/CsBe/Modul 404/Exercises', with a note below it stating 'Project will be created in: ~/.Git/CsBe/Modul 404/Exercises/06 HelloWorldFX'. There is an unchecked checkbox for 'Create Git repository'. Under 'Language', 'Java' is selected. Under 'Build system', 'Maven' is selected. The 'Group' field contains 'ch.csbe.modul404'. The 'Artifact' field contains 'HelloWorldFX' and is highlighted with a blue border. The 'JDK' dropdown shows '18 version 18'.

2. Klicken Sie auf **Create** ohne weitere Bibliotheken hinzuzufügen.
3. Analysieren Sie die beiden vom Template erstellten Klassen **HelloApplication** und **HelloController**.
 - a. Sollten Sie den Code nicht verstehen, wenden Sie sich an die Lehrperson.
4. Benennen Sie die beiden vorhandenen Klassen um in: **HelloWorld** und **HelloWorldController**
5. Benennen Sie die hello-view.fxml um in **HelloWorld.fxml**
6. Öffnen Sie die HelloWorld.fxml im SceneBuilder und löschen Sie die vorhandene VBox im Tab Hierarchie.
 - a. Platzieren Sie stattdessen einen Anchor Pane.
7. Gestalten Sie ein GUI mit einem Button dem Sie im Objekttab Code eine fx:Id geben z.B. btnHello und ein Label, dem Sie ebenfalls eine Id vergeben z.B. lblHello
8. Wechseln Sie in die Klasse HelloWorldController und fügen Sie folgenden Code ein:

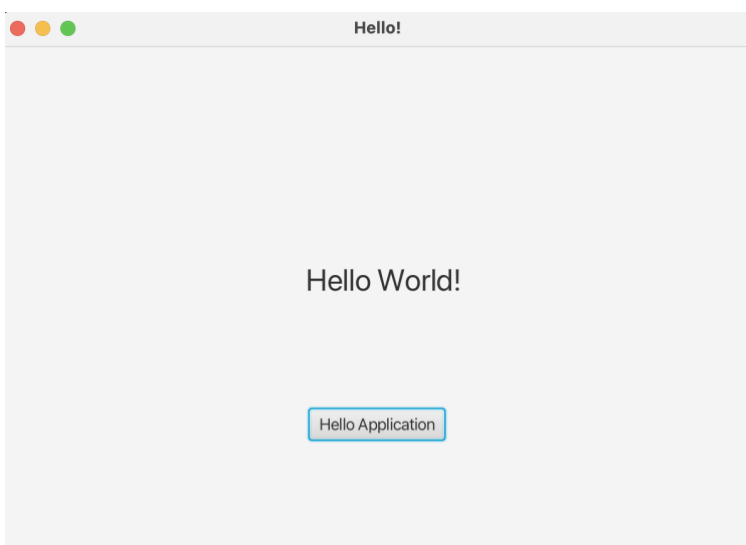
```
public class HelloWorldController {  
    @FXML  
    private Label lblHello;  
  
    @FXML  
    public void onHelloButtonClick() {  
        lblHello.setText("Hello World!");  
    }  
}
```

9. Wechseln Sie zurück in den SceneBuilder und verdrahten Sie die Methode onHelloButtonClick mit dem btnHello.
 - a. Fügen Sie die Controller-Klasse unter dem Tab Controller hinzu.
 - b. Klicken Sie auf den Button und klicken Sie im Objekttab Code auf OnAction und wählen Sie die Methode aus dem Controller aus.
 - c. Speichern Sie ihre Anpassungen.
10. Wechseln Sie zurück in die IDE und löschen Sie das Fenstergrößenargument aus der folgenden Codezeile:
 - a. `Scene scene = new Scene(fxmlLoader.load(), 320, 240);`

Ihre Applikation sollte nun etwa so aussehen, wenn Sie diese starten:



Und wenn Sie auf den Button Hello Application drücken, sollte sie in etwa wie folgt aussehen:



4 Aufgabe: Startleiste für das Modul 404

Nun da wir unsere erste JavaFX Applikation erstellt haben, ist es nun an der Zeit eine für uns nützliche Applikation zu erstellen. Im Modul 404 arbeiten Sie viel mit IntelliJ, dem SceneBuilder und wahrscheinlich der Google-Suche und StackOverflow, wäre es nicht nützlich diese Programme in einer Schnellleiste immer zur Hand zu haben?

Ziel der Aufgabe:

Erstellen Sie eine handgemacht Startleiste mit 5 Knöpfen (sie dürfen natürlich auch mehr einbauen):

- IntelliJ
- SceneBuilder
- Google-Search
- StackOverflow
- Close

Verlinken Sie die Buttons mit der jeweiligen Applikation bzw. der entsprechenden URL.

Vorgehen:

Projekt-Setup:

1. Erstellen Sie ein neues JavaFX-Projekt in ihrem Repository und benennen Sie es mit **07 LaunchBarFX**
 - a. Klicken Sie auf **Create** ohne weitere Bibliotheken hinzuzufügen.
2. Benennen Sie die beiden vorhandenen Klassen um in: **LaunchBar** und **LaunchBarController**
3. Benennen Sie die hello-view.fxml um in **launchBar.fxml**
4. Gehen Sie in das **Teams Modul404 → Kursunterlagen → 04 Sonstiges → LaunchBar Images** und kopieren Sie die darin enthaltenen Bilder in ihr Projekt unter **resources/package**.

UI-Setup:

1. Öffnen
2. Sie ihre launchBar.fxml in SceneBuilder
3. Löschen Sie die VBox und alle Ihre Inhalte unter dem Tab Hierarchy.
4. Wählen Sie unter dem Tab Containers die HBox aus und ziehen Sie diese auf die Arbeitsfläche.
 - a. Unter dem Objekttab Properties wählen Sie für Alignment **CENTER** aus.
 - b. Unter dem Objekttab Layout wählen Sie für die Breite 360 px und für die Höhe 70 px aus.
 - c. Im gleichen Tab setzen Sie das Spacing auf 10 px.
5. Fügen Sie nun 5 Buttons ein, diese befinden sich unter dem Tab Controls.
 - a. Für die Buttons setzen Sie die Breite und die Höhe auf jeweils 64 px.
 - b. Geben Sie den Buttons eine eindeutige Bezeichnung, damit Sie diese später im Code ansprechen können. Objekttab Code → fx:id = **btn1** (bis Button 5)
6. Ihre Benutzeroberfläche sollte nun wie folgt aussehen:

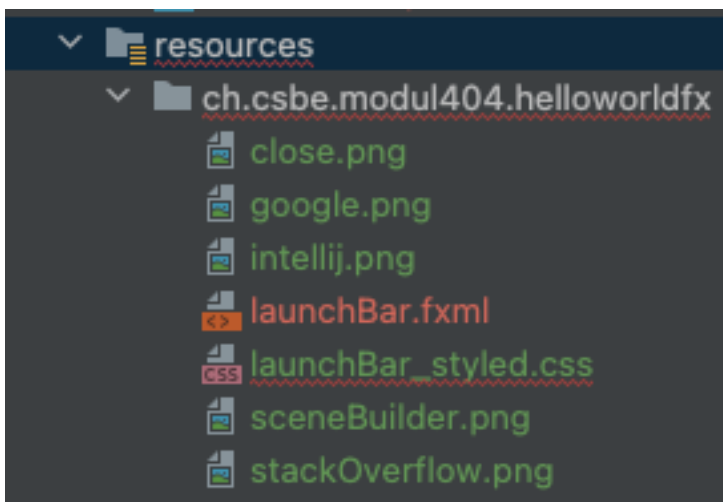


7. Entferne die Argumente für die Scenegrösse im Code auf Zeile 13.
8. Setzen Sie den korrekten Titel für Ihre Stage auf Zeile 15 (z.B. LaunchBar M404).
9. Vergessen Sie nicht ihre Anpassungen regelmässig zu speichern.

UI-Styling:

Die Benutzeroberfläche sieht schonmal zweckmässig aus. Damit wir jedoch schöne Buttons für unsere Applikationen sowie Webseiten designen können, benötigen wir zusätzliches Styling. Zum Glück supportet JavaFX die populäre Styling-Sprache CSS. Um CSS einzubinden, müssen wir eine .css-Datei erstellen und diese in unsere Applikation einbinden.

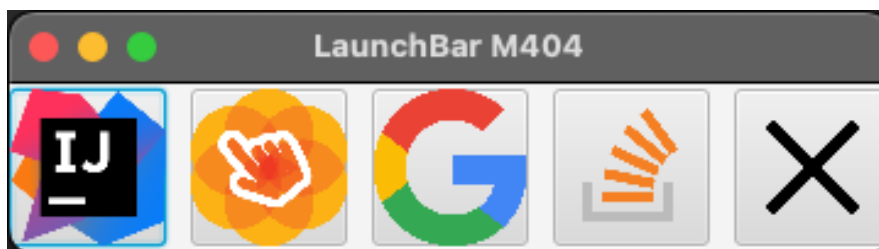
1. Gehen Sie zurück ins IntelliJ und erstellen Sie unter dem Verzeichnis **resources/package** eine css Datei (launchBar_styled.css).



2. Definieren Sie folgende CSS-Klassen:

```
.btn1 {  
    -fx-background-image: url(intellij.png);  
    -fx-background-size: 64.0, 64.0;  
    -fx-background-repeat: no-repeat;  
    -fx-background-position: center;  
}  
  
.btn2 {  
    -fx-background-image: url(sceneBuilder.png);  
    -fx-background-size: 64.0, 64.0;  
    -fx-background-repeat: no-repeat;  
    -fx-background-position: center;  
}  
  
.btn3 {  
    -fx-background-image: url(google.png);  
    -fx-background-size: 64.0, 64.0;  
    -fx-background-repeat: no-repeat;  
    -fx-background-position: center;  
}  
  
.btn4 {  
    -fx-background-image: url(stackOverflow.png);  
    -fx-background-size: 64.0, 64.0;  
    -fx-background-repeat: no-repeat;  
    -fx-background-position: center;  
}  
  
.btn5 {  
    -fx-background-image: url(close.png);  
    -fx-background-size: 64.0, 64.0;  
    -fx-background-repeat: no-repeat;  
    -fx-background-position: center;  
}
```

3. Zurück im SceneBuilder müssen wir nun die Styles den Buttons hinzufügen, dafür muss das Stylesheet erstmals referenziert werden. Klicke auf einen Button navigiere im Objekttab auf Properties und füge anschliessend das Stylesheet hinzu. Unter Style Class kannst du nun die entsprechende Klasse auswählen.
 - a. Führe die Anpassung für alle Buttons aus.
4. Speichere dein Benutzerinterface und schau dir in der IDE, das dazugehörige XML an welches in der Datei **launchBar.fxml** für dich erstellt wurde.
 - a. Hast du Fragen zum XML, wende dich an die Lehrperson.
5. Dein Projekt sollte jetzt wie folgt aussehen:



Handle Events - LaunchBarController:

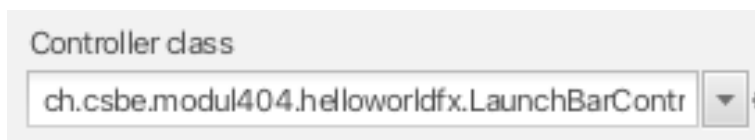
Die Benutzeroberfläche sieht schonmal recht schön aus und erfüllt die UI-Anforderungen für unsere Applikation. Jetzt müssen die Knöpfe noch mit dem Programm, der Java-Applikation, verdrahtet werden. Hierfür haben wir schon eine Klasse, nämlich **LaunchBarController**. Zur Erinnerung, Klassen sollten immer nur eine Aufgabe haben und diese möglichst gut erfüllen (Prinzip: Encapsulation – Separation of Concerns).

Dafür müssen sog. Events (als wenn z.B. ein Knopf gedrückt wird) verarbeitet oder gehandelt werden. Events sind meistens Userinteraktionen wie z.B.:

- Linke Maustaste gedrückt
- Rechte Maustaste gedrückt
- Taste «x» gedrückt
- Drag and Drop
- Hover
- ...

Um ihre Benutzeroberfläche mit der Java-Applikation zu verdrahten und Klick-Events auf die Buttons zu handeln gehen Sie wie folgt vor:

1. Fügen Sie den Controller im Tab Controller hinzu.

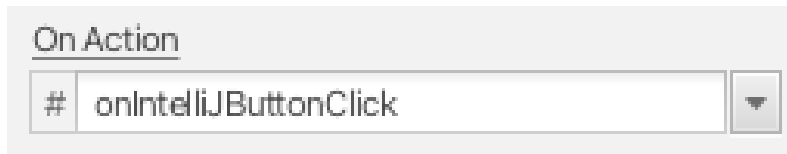


2. Navigieren Sie zurück ins IntelliJ, öffnen Sie die Klasse **LaunchBarController** und beginnen Sie die einzelnen Methoden zu schreiben, die über den jeweiligen Button-Klick-Event ausgeführt werden sollen.

```
public void onIntelliJButtonClick(ActionEvent event) {  
}  
  
public void onSceneBuilderClick(ActionEvent event) {  
}  
  
public void onGoogleClick(ActionEvent event) {  
}  
  
public void onStackOverflowClick(ActionEvent event) {  
}  
  
public void onCloseClick(ActionEvent event) {  
    System.exit(0);  
}
```

3. Vom Modul 403 kennen Sie bereits den Befehl, mit welchem Sie eine Java-Applikation schliessen können, diesen **System.exit(0)**; können wir bereits hinzufügen und testen.

4. Um die Methoden mit den Buttons im SceneBuilder zu verbinden, wechseln Sie zurück in den SceneBuilder und klicken auf den Button. Klicken Sie im Objekttab unter Code auf das DropDown bei OnAction und wählen Sie die entsprechende Methode aus.



5. Wie Sie wissen, ist eine der grossen Vorteile von Java, dass die Programme plattformunabhängig laufen. Dies gilt jedoch nur für die Programmiersprache, Betriebssystemspezifische Eigenheiten wie z.B. Pfadangaben müssen durch den Entwickler entsprechend abgefangen und behandelt werden.
 - a. Überlegen Sie sich welche Betriebssystemspezifische Herausforderungen könnten auf uns warten, wenn wir die Applikation Windows und Mac unterstützen soll?
 - b. Überlegen Sie sich welche Methoden sie implementieren müssen, damit Sie die gewünschten Funktionalitäten anbieten können?
 - i. Die Applikation muss das Betriebssystem bestimmen können.
 - ii. Die Applikation muss ein Programm auf einem Mac-OS starten können.
 - iii. Die Applikation muss ein Programm auf einem Win-OS starten können.
 - iv. Die Applikation muss eine Webseite auf einem Mac-OS aufrufen können.
 - v. Die Applikation muss eine Webseite auf einem Win-OS aufrufen können.
6. Um das Betriebssystem herauszufinden, verwenden wir eine spezielle Methode, den sog. **Konstruktor**. Diesen werden Sie zu einem späteren Zeitpunkt noch besser kennenlernen, sollten Sie ihn noch nicht im Unterricht angetroffen haben.

```
private String os;  
  
public LaunchBarController() {  
    String os = System.getProperty("os.name").toLowerCase();  
  
    if (os.contains("win"))  
        this.os = "win";  
    if (os.contains("mac"))  
        this.os = "mac";  
    if (os.contains("nix") || os.contains("nux"))  
        throw new RuntimeException("Linux is not supported, sorry :(.");  
}
```

7. Folgende Methoden implementieren das betriebssystemabhängige Starten von Applikationen:
 - a. **Hinweis:** Der Code konnte erst auf einem Mac getestet werden. Sollte er nicht auf Anhieb funktionieren, versuchen Sie dies als Herausforderung selbständig zu lösen. Sollten Sie Unterstützung benötigen, melden Sie sich bei der Lehrperson.


```
private void startApplicationOnMac(String[] command) {

    ProcessBuilder processBuilder = new ProcessBuilder(command);
    processBuilder.directory(new File(System.getProperty("user.home")));

    try {
        processBuilder.start();
    } catch (IOException exception) {
        exception.printStackTrace();
    }
}

private void startApplicationOnWindows(String command) {
    ProcessBuilder processBuilder = new ProcessBuilder(command);
    try {
        processBuilder.start();
    } catch (IOException exception) {
        exception.printStackTrace();
    }
}
```

8. Die beiden Methoden startApplicationOnMac und startApplicationOnWindows sind sog. Implementierungsdetails und werden deshalb nach aussen hin versteckt (private), entsprechend dem zweiten Grundprinzip der Abstraction.
9. Nun müssen noch die beiden Methoden zum betriebssystemabhängigen Aufrufen der Webseiten implementiert werden, eine mögliche Lösung sieht wie folgt aus:

```
private void openWebSiteOnMac(String[] command) {
    Runtime rt = Runtime.getRuntime();
    try {
        rt.exec(command);
    } catch (IOException exception) {
        exception.printStackTrace();
    }
}

private void openWebSiteOnWindows(String url) throws Exception {
    Desktop desktop = Desktop.getDesktop();
    desktop.browse(new URI(url));
}
```

10. Als letztes folgt die Implementierung der Event-Handler, also der Methoden, die direkt mit dem GUI verknüpft sind. Eine Implementierung dieser könnte wie folgt aussehen:

```

public void onIntelliJButtonClick(ActionEvent event) {
    if (os.equals("mac")) {
        String[] command = {"open", "/Applications/IntelliJ IDEA.app"};
        startApplicationOnMac(command);
    } else {
        String command = "\"C:\\Program Files\\JetBrains\\IntelliJ IDEA Community Edition 2022.1\\bin\\idea64.exe\"";
        startApplicationOnWindows(command);
    }
}

public void onSceneBuilderClick(ActionEvent event) {
    if (os.equals("mac")) {
        String[] command = {"open", "/Applications/SceneBuilder.app"};
        startApplicationOnMac(command);
    } else {
        String command = "\"C:\\Users\\michael\\AppData\\Local\\SceneBuilder\\SceneBuilder.exe\"";
        startApplicationOnWindows(command);
    }
}

public void onGoogleClick(ActionEvent event) {
    String url = "https://google.com";
    if (os.equals("mac")) {
        String[] command = {"open", url};
        openWebSiteOnMac(command);
    } else {
        try {
            openWebSiteOnWindows(url);
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

public void onStackOverflowClick(ActionEvent event) {
    String url = "https://stackoverflow.com";
    if (os.equals("mac")) {
        String[] command = {"open", url};
        openWebSiteOnMac(command);
    } else {
        try {
            openWebSiteOnWindows(url);
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
}

```

11. Diese Methoden müssen zwingend public sein, da Sie von ausserhalb der Klasse, nämlich aus der JavaFX-Applikation, der LaunchBar Klasse aufgerufen werden.
12. Starten Sie die Applikation und überprüfen Sie die Funktionalität der Buttons, diese sollten nun wie gewünscht funktionieren.

UI-Feinschliff:

Unsere Applikation ist nun funktional gesehen komplett. Nun geht es darum dem Benutzerinterface den letzten Feinschliff zu geben. Zum einen wollen wir den Hintergrund transparent machen und zum anderen die Applikation, nach dem Start, in der oberen Mitte unseres Bildschirms positionieren.

1. Da es sich hierbei um Styling handelt müssen wir unser Stylesheet ergänzen. Damit dem `rootNode` aus dem SceneBuilder in der Applikation ansprechbar wird, muss diesem eine ID zugeteilt werden.
2. Navigieren Sie in den SceneBuilder und klicken Sie auf das Layout, klicken Sie im Objekttab Code in die Textbox `fx:id` und vergeben Sie die ID: **rootNode**
3. Im StyleSheet können Sie nun das Layout sowie die Buttons transparent machen. Fügen Sie folgende Klassen hinzu:

```
.rootNode {  
    -fx-background-color: transparent;  
}  
  
.button {  
    -fx-background-color: transparent;  
}
```

4. Vergessen Sie nicht diese im SceneBuilder im Objekttab unter Properties StyleSheet einzubinden.



5. Als nächsten Schritt wird die Scene sowie die Stage transparent gestellt. Da diese beiden ebenen oberhalb des RootNodes liegen, müssen diese im Code angepasst werden.
6. Wechseln Sie ins **IntelliJ** und in die Klasse **LaunchBar**. Ergänzen Sie die LaunchBar mit folgenden Codezeilen:
 - a. `scene.setFill(Color.TRANSPARENT);`
 - b. `stage.initStyle(StageStyle.TRANSPARENT);`
7. Als letzten Schritt müssen wir dafür sorgen, dass die Applikation am oberen Bildschirmrand in der Mitte ausgerichtet wird. Dafür wird wie folgt vorgegangen:
 - a. Erfassung der der Bildschirmgröße in einem Rechteck mithilfe der Screen-Klasse:
 - i. `Rectangle2D screenSize = Screen.getPrimary().getVisualBounds();`
 - b. Für die Ausrichtung benötigt unsere Applikation nun die X sowie die Y Koordinate. Standardmässig richtet sich die Applikation am oberen linken Bildschirmrand aus. Für eine Positionierung in der Mitte, müssen folgende Codezeilen hinzugefügt werden:
 - i. `stage.setX(screenSize.getWidth()/2 - 180);`
 - ii. `stage.setY(20.0);`

```
public class LaunchBar extends Application {  
  
    Rectangle2D screenSize = Screen.getPrimary().getVisualBounds();  
  
    @Override  
    public void start(Stage stage) throws IOException {  
        FXMLLoader fxmlLoader = new  
FXMLLoader(LaunchBar.class.getResource("launchBar.fxml"));  
        Scene scene = new Scene(fxmlLoader.load());  
        scene.setFill(Color.TRANSPARENT);  
        stage.initStyle(StageStyle.TRANSPARENT);  
        stage.setX(screenSize.getWidth()/2 - 180);  
        stage.setY(20.0);  
        stage.setScene(scene);  
        stage.show();  
    }  
  
    public static void main(String[] args) {  
        launch();  
    }  
}
```

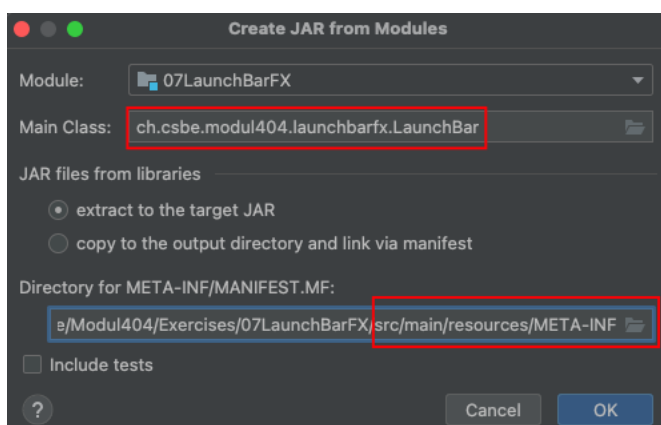
8. Herzlichen Glückwunsch, die Applikation ist nun vollständig und sollte wie folgt aussehen:



5 Bereitstellen von ausführbaren Java-Programmen

Um ein abgeschlossenes Java-Projekt für die Endbenutzer verfügbar zu machen, muss dies in einem ausführbaren Format bereitgestellt werden. Um eine ausführbare Datei aus einem Java-Projekt zu erstellen, gehen Sie in IntelliJ wie folgt vor:

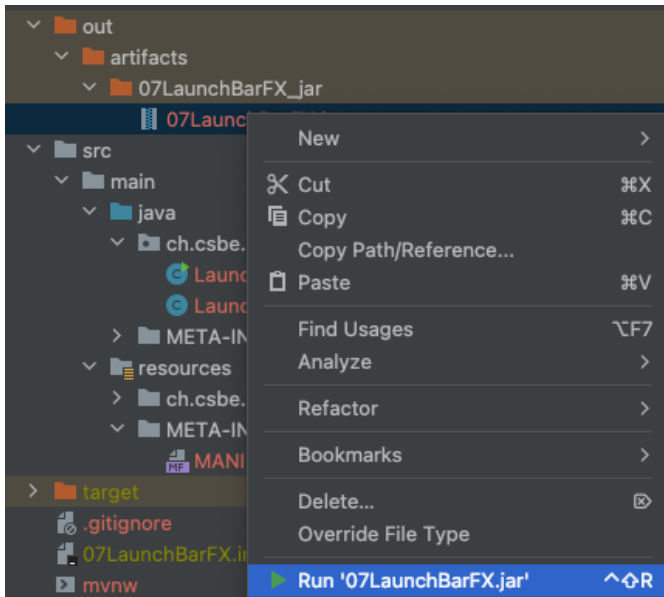
1. Klicken Sie im IntelliJ auf den Menüpunkt **Build** und klicken Sie dann auf **Build Project**.
2. Klicken Sie auf den Menüpunkt **File** → **Project Structure** → **Artifacts** → **+** → **Jar** → **Modules with dependencies** → Füllen Sie die benötigten Angaben aus.
3. Beachten Sie, dass Sie die korrekten Einstellungen für die **Main-Klasse** und das **MANIFEST.MF** vornehmen.



4. Schauen Sie unter **Output directory** wo ihr .jar-File anschliessend erstellt wird.
5. Klicken Sie auf OK und auf Apply und anschliessend OK.
6. Klicken Sie im Menüpunkt auf **Build → Build Artifacts... → 07 LaunchBarFX.jar → Build**.
7. Nun finden Sie eine ausführbare .jar-Datei in ihrem Output directory.

5.1 JVM Options für JavaFX Applikationen

Wenn Sie im IntelliJ Ihre frisch gebackene .jar-Datei auswählen öffnen, werden Sie feststellen, dass diese nicht gestartet werden kann.



```
/Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home/bin/java -Dfile.encoding=UTF-8
Error: JavaFX runtime components are missing, and are required to run this application
```

Dies liegt daran, dass zum Starten einer gepackten JavaFX-Applikation gewisse Optionen für die Java-Virtual-Machine mitgegeben werden müssen, nachzulesen unter «**JavaFX and IntelliJ → Non-modular from IDE → 4. Add VM options**» in der [offiziellen Dokumentation von JavaFX](#).

Linux/Mac

Windows

```
--module-path "%path%\to\javafx-sdk-17.0.1\lib" --add-modules javafx.controls,javafx.fxml
```

```
--module-path "%path%\to\javafx-sdk-17.0.1\lib" --add-modules javafx.controls,javafx.fxml
```

Um diese Optionen für den Start der .jar-Datei mitzugeben, gehen Sie in IntelliJ wie folgt vor:

1. Klicken Sie in der Menüleiste auf **Run → Edit configurations...** → und fügen Sie die Optionen im Feld **VM options** ein.
 - a. Vergessen Sie nicht die Pfadangaben zum JavaFX für ihren Rechner anzupassen. Fall Sie JavaFX noch nicht heruntergeladen haben, können Sie dies hier tun, oder sie kopieren die Software aus dem Teams → Moul404 → Kursmaterialien → 07 Software → openifx-18.0.1_windows-x64....

2. Wenn Sie die .jar-Datei nun aus IntelliJ starten, ist die Fehlermeldung verschwunden und die Applikation startet wie gewünscht.

Jetzt ist bekannt welche VM-Optionen benötigt werden. Nun müssen diese jedoch beim Starten der Applikation, ausserhalb von IntelliJ mitgegeben werden. Um ein installierbares Paket zu erstellen, verwenden wir das seit Java 14 im JDK enthaltene Tool jpackage.


Um eine ausführbare Datei (.exe oder .msi) in Windows zu erstellen, gehen Sie wie folgt vor:


1. Öffnen Sie PowerShell und navigieren Sie in das Projektverzeichnis 07LaunchBarFX.
2. Laden Sie sich ein passendes .ico von der Seite <https://icon-icons.com/> herunter und speichern Sie dieses in ihrem Projekt im resource Folder zu den anderen Bildern.
3. Geben Sie nun folgenden Befehl ein (beachten Sie, dass Sie den Pfad zu der Installation ihrer JavaFX Module angeben – und die anderen gelb markierten Stellen):

```
jpackage.exe --name "LaunchBarFX M404" --type exe --input . --dest
.\out\artifacts\07LaunchBarFX_jar\ --main-jar
.\out\artifacts\07LaunchBarFX_jar\07LaunchBarFX.jar --win-shortcut --win-menu --
java-options "--module-path C:\JavaFX\javafx-sdk-18.0.1\lib --add-modules
javafx.controls,javafx.fxml" --icon
"..\src\main\resources\ch\csbe\modul404\launchbarfx\launch.ico"
```

* Informationen zum ausgeführten Command finden Sie im [offiziellen User-Guide](#) von Oracle.

4. Sie erhalten nun eine ausführbare .exe Datei in ihrem angegebenen Verzeichnis:
"C:\Modul404\Exercises\07LaunchBarFX\out\artifacts\07LaunchBarFX_jar\LaunchBarFX M404-1.0.exe"

 07LaunchBarFX.jar

 LaunchBarFX M404-1.0.exe

5. Installieren Sie nun die Software, anschliessend erhalten Sie eine Verknüpfung auf Ihrem Desktop und einen Shortcut in Ihrer Menüleiste.

