



Arbeitsschritte der Programmerstellung:

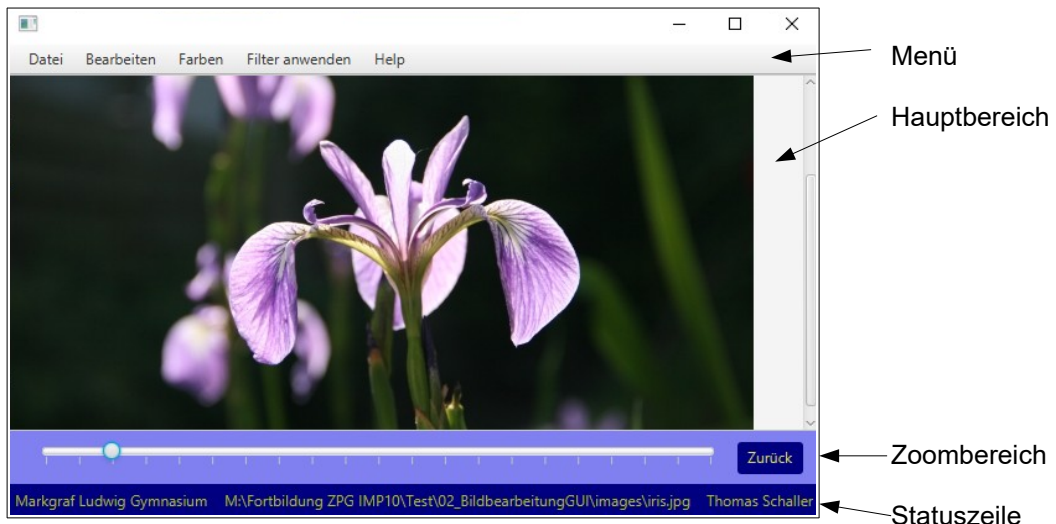
Nachdem die Algorithmen zur Bildbearbeitung implementiert und getestet wurden, kann nun die Programmoberfläche (Graphical User Interface) erstellt und mit den Algorithmen verknüpft werden. Daraus ergeben sich die drei Schritte:

1. Implementation und Testen der Algorithmen (**Model**)
2. Gestalten der Oberfläche mit einem GUI-Editor: hier Gluon Scene Builder (**View**)
3. Verknüpfen der GUI mit der Oberfläche: Implementation von Aktionen (**Controller**)

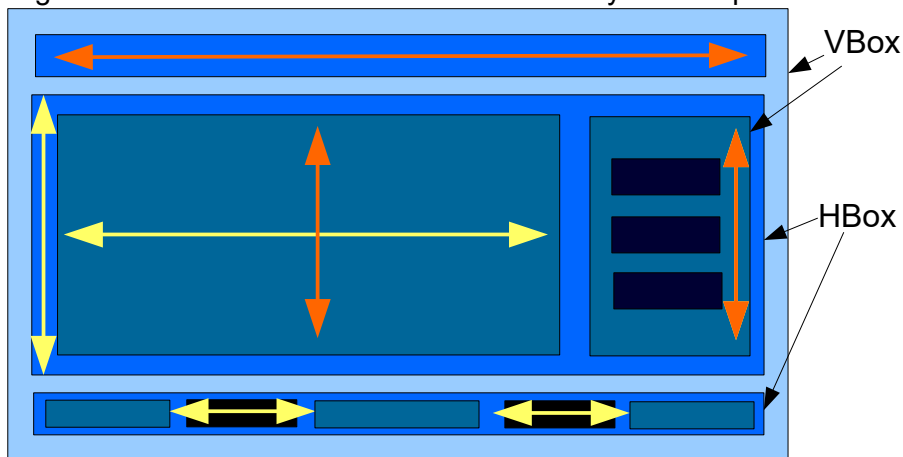
Die Trennung eines Programmierprojekts in diese drei Teile nennt man auch **Model-View-Controller (MVC)** Konzept und ist für größere Projekte Pflicht, da es die Übersicht erhöht. Die Interaktion zwischen den drei Teilen vereinfachen wir aber hier in der Schule gegenüber dem "echten" MVC-Konzept.

Gestaltung der Oberfläche

Am Ende soll die Oberfläche in etwa so aussehen:



Das Layout der Seite sollte so gestaltet werden, dass es auch bei einer Änderung der Größe des Programmfensters noch ansprechend aussieht. Daher muss man sich überlegen, welche Bereiche sich ausdehnen sollen und welche ihre Größe beibehalten sollen. Grundsätzlich besteht das Programm aus einer Reihe verschachtelter Layout-Komponenten. Die wichtigsten





sind VBox (ordnet die untergeordneten Elemente vertikal an), HBox (ordnet die untergeordneten Elemente horizontal an) und AnchorPane (freie Positionierung der untergeordneten Elemente). Bei den untergeordneten Elementen kann man im Layout festlegen, ob das Element so groß sein soll, wie es notwendig ist, ob es so groß sein soll wie das übergeordnete Element (oranger Pfeil) oder den zur Verfügung stehenden Platz füllen soll (gelbe Pfeile).

Vorbereiten des Gluon Scene Builders

1. Video: *gui_erstellen_0.mp4*



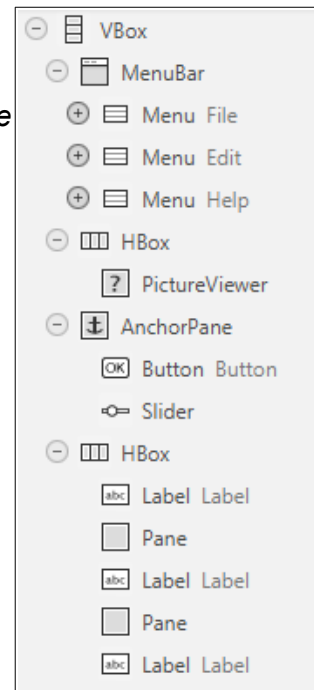
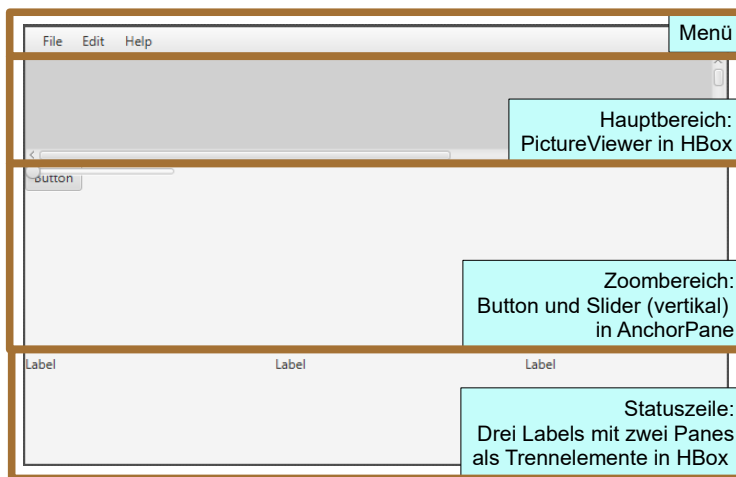
Füge die beiden Elemente *PictureViewer* und *NumberField* aus der Datei *imtkomponenten.jar* dem *SceneBuilder* hinzu.

Elemente hinzufügen

2. Video: *gui_erstellen_1.mp4*



Füge neue Elemente zu der Oberfläche hinzu, so dass die GUI wie gezeigt aussieht. Die Position auf dem Bildschirm ist zunächst noch egal.

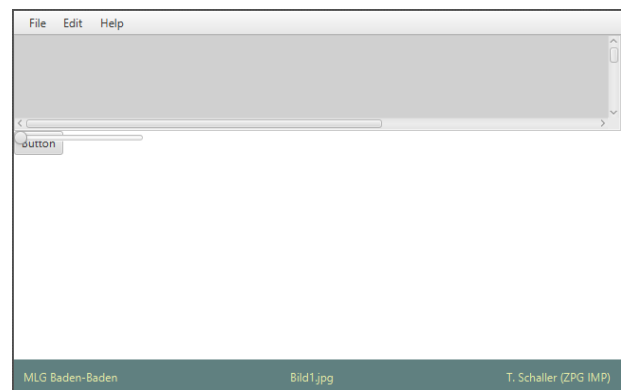


Statuszeile

Die Statuszeile enthält drei Labels für den Schulnamen, für den Dateinamen des angezeigten Bildes und für deinen eigenen Namen.

Damit die Labeltexte nicht so am Rand kleben, wird ein Abstand zum Rand des HBox-Elements eingefügt (Padding). Auch zwischen den Elementen soll etwas Abstand sein (Spacing).

Die Pane-Elemente sind dazu da, die Labeltexte über die ganze Breite zu verteilen. Daher müssen sie den ganzen zur Verfügung stehenden Platz ausfüllen (HGrow = ALWAYS).



Lange Texte auf den Labels werden abgeschnitten, da das Element zu klein ist. Daher muss die minimale Größe des Elements der bevorzugten Größe (Preferred Size) entsprechen. Um die Optik zu verbessern, kann man außerdem die Hintergrundfarbe der HBox und die Textfarbe der Labels anpassen.

3. Video: *gui_erstellen_2.mp4*




Nimm diese Änderungen an der Statuszeile vor.



Horizontale Aufteilung des zur Verfügung stehenden Platzes

In der umgebenden VBox ist das Menü, zwei HBoxen und eine Anchorpane enthalten.

4. Stelle die Option VGrow im Layout so ein, dass sich nur die erste HBox (mit dem  PictureViewer) ausdehnt, die anderen Elemente sich aber nicht ausdehnen.

Zoombereich

Der Zoombereich ist eine AnchorPane. Daher können die Elemente hier mit der Maus an die gewünschten Positionen gezogen werden. Man muss sich dann noch überlegen, an welchen Rändern sie verankert (anchor = Anker) werden sollen. Ändert sich die Größe der AnchorPane, bleiben die Abstände zu diesen Rändern erhalten. Verankert man das Element nur auf einer Seite, verschiebt sich das Element auf der AnchorPane. Verankert man linken und rechten Rand, passt es seine Größe an. Gleiches gilt für den oberen und unteren Rand.

Über den Slider soll das angezeigte Bild mit einem Faktor zwischen 0,01 und 10 gezoomt werden. Startwert ist Faktor 1. Es soll jeweils nach 0,5 ein Strich (Tick Mark) angezeigt werden. Dazwischen werden 4 weitere kleinere Striche (Minor Tick) angezeigt.

Beim Button muss der Text auf "Zurück" geändert werden. Dieser Button soll die letzte Änderung am Bild rückgängig machen.

5. Video: *gui_erstellen_3.mp4*.
 Passe den Zoombereich nach den Vorgaben an. Du kannst auch hier wieder die Farbeinstellungen anpassen.

Menüzeile

Ein Menü besteht aus einem MenuBar-Element, das mehrere horizontal angeordnete DropDown-Menüs (Menu-Elemente) enthält. Diese enthalten einzelne Menüpunkte (MenuItem-Element) und Trennstriche (SeparatorMenuItem-Element).

Folgende Struktur soll das Menü haben. Mit der rechten Maustaste können Elemente dupliziert werden, um das Eingeben zu beschleunigen.

Menüstruktur:

Datei	Bearbeiten	Farben	Effekte	Kunst	Info
Bild öffnen...	Zurück	Invertieren	Weichzeichnen	Relief	Über mein Programm...
Bild speichern ...	Drehe links	Graustufen (Minimum)	Schärfen		
Beenden	Drehe rechts	Graustufen (Maximum)	Finde horizontale Kanten		
	Spiegle horizontal	Graustufen (Mittelwert)	Finde vertikale Kanten		
	Spiegle vertikal	Graustufen (Natürlich)	Finde alle Kanten		
		Tausch Rot-Grün			
				
		Nur Rotkanal			
				

Bei den Oberpunkten wie "Datei" kann das "Mnemonic Parsing" aktiviert werden. Wenn man als Text dann "_Datei" eingibt, wird das "D" als Kürzel für das "Datei"-Menü verwendet. Die Tastenkombination Alt+D öffnet dann das Menü. Für einzelne Menüpunkte können Tastenkürzel definiert werden. Üblich sind Ctrl+O für "Öffnen", Ctrl+S für "Speichern" und Ctrl+Q für "Beenden". Weitere können frei gewählt werden: z.B. Ctrl+L für "Drehe links".



6. Video: *gui_erstellen_4.mp4*.



Passe das Menü wie oben gezeigt an. Du kannst weitere Menüpunkte ergänzen, um mehr Funktionen zu ermöglichen.

GUI starten

Der SceneBuilder speichert die Oberfläche als FXML-Datei. Diese kann durch ein einfaches Java-Programm geladen werden und automatisch angezeigt werden. Dazu wird die FXML in dem Unterordner "view" des BlueJ-Projekts "02_bildbearbeitungsprogramm" gespeichert.

7. Öffne das BlueJ-Projekt "bildbearbeitungsprogramm". Passe in der Klasse



"BildbearbeitungGUI" den Namen der FXML-Datei an.

Starte die GUI mit der rechten Maustaste -> "Run JavaFX Application".

Controller

Alle GUI-Elemente müssen nun noch mit Leben versehen werden. Dafür ist der Controller verantwortlich. Zunächst gibt man im Scene Builder an, wie man die Controller-Klasse nennen möchte: hier einfach "Controller".

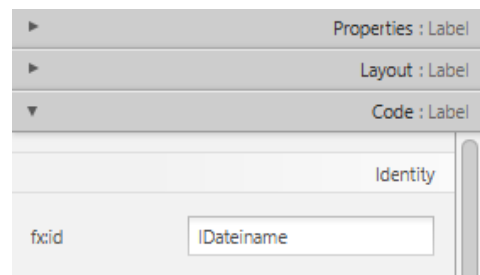
Ereignisbehandlung für Buttons und Menüpunkte

Wenn ein Menüpunkt oder ein Button gewählt wird, wird eine Methode im Controller aufgerufen. Im Menüpunkt Code -> On Action kann man festlegen, wie diese Methode heißen soll. Da es sich um Methodennamen handelt, sollten die Namen mit Kleinbuchstaben beginnen und im CamelCase geschrieben sein. Für die Übersicht ist es gut, Button-Methoden mit einem kleinen "b" beginnen zu lassen und Menü-Methoden mit einem kleinen "m".

z.B. bZurueck oder mDreheLinks

Namen für Kontrollelemente

Jedes Element, das sich während der Laufzeit des Programms ändern kann (z.B. anderer Text bei einem Label, Sliderposition usw.) benötigt einen eindeutigen Namen. Dieser wird im Controller als Name für die Variable benutzt. Man stellt ihn bei der Eigenschaft "fxid" ein.



8. Video: *gui_erstellen_5.mp4*.



Trage den Controllernamen ein und lege für den "Zurück"-Button und jeden Menüpunkt einen Methodennamen fest. Benenne die Elemente mit "slZoom" (Slider zum Zoomen), "IDateiname" (Label für Dateinamen), "viewer" (ImageViewer) und "hauptbereich" (HBox des ImageViewers).

Implementation des Controllers

Im Scene Builder erhält man über Menü View -> Show Sample Controller Skeleton einen Beispiel-Controller, der mit Copy-Paste in BlueJ (neue Klasse -> Name "Controller") übernommen wird. Die Import-Anweisungen am Anfang des Codes müssen noch etwas angepasst werden, damit alle notwendigen Bibliotheken eingebunden werden.



```
import imp.*;                // Alles aus IMP Unterordner
import javafx.fxml.*;         // FXML Definitionen
import javafx.scene.control.*; // Jedes Control-Element
import javafx.scene.layout.*; // Alle Layout-Definitionen
import javafx.event.*;        // Button und Menu-Events
import javafx.stage.*;        // Dateiöffnen / Speichern-Dialog
import java.io.File;           // Dateihandling
```

Man sieht, dass im Code für jeden Button und jeden Menüpunkt eine eigene Methode eingefügt wurde. Diese wird ausgeführt, wenn man den Button bzw. Menüpunkt anklickt. Für jedes benannte Kontroll-Element wurde ein Attribut mit gleichem Namen am Anfang der Klasse eingefügt. Über diesen Namen kann es angesprochen werden.

9. Video: *gui_erstellen_6.mp4*.



Erstelle die Controller-Klasse und passe die Import-Anweisungen an. Teste, ob sich das Programm weiterhin ausführen lässt.

Erste Aktion: Datei öffnen

Wählt man "Bild öffnen", soll ein Dialogfenster aufgehen, in dem das Bild ausgewählt werden kann. Diesen Dialog gibt es schon fertig in der Java-Klasse `FileChooser`. Um ihn zu benutzen, deklarieren wir nach den FXML-Attributen ein eigenes Attribut `dateidialog`.

```
// Eigene Attribute
private FileChooser dateidialog;
```

Außerdem fügen wir eine neue Methode `public void initialize()` ein. Diese wird von JavaFX aufgerufen, nachdem die FXML Elemente erzeugt und verknüpft wurden.

```
public void initialize() {
    dateidialog = new FileChooser();
    dateidialog.setInitialDirectory(new File("images"));
}
```

Damit wird ein neuer Dialog erzeugt (aber noch nicht angezeigt) und das Startverzeichnis auf das Unterverzeichnis "images" gesetzt.

In der "mBildOeffnen"-Methode kann der Dialog nun genutzt werden, um das Bild auszuwählen und falls der Dialog nicht abgebrochen wurde (`file != null`), wird das Bild geladen und im `ImageViewer`-Element "viewer" angezeigt. Außerdem wird das Label "lDateiname" angepasst.


```
@FXML
void mBildOeffnen(ActionEvent event) {
    File file = dateidialog.showOpenDialog(null);
    if (file != null) {
        Picture neuesBild = new Picture(file.getAbsolutePath());
        viewer.setImage(neuesBild, true);
        lDateiname.setText(file.getAbsolutePath());
    }
}
```

Speichern funktioniert auf die gleiche Weise. Wurde ein Dateiname gewählt, wird der "viewer" nach dem aktuellen Bild befragt und dieses Bild unter dem gewählten Dateinamen gespeichert.



```
@FXML
void mBildSpeichern(ActionEvent event) {
    File file = dateidialog.showSaveDialog(null);
    if (file != null) {
        Picture aktuellesBild = viewer.getImage();
        aktuellesBild.save(file.getAbsolutePath());
        lDateiname.setText(file.getAbsolutePath());
    }
}
```

10. Video: *gui_erstellen_7.mp4*.

 *Füge das Datei-Öffnen und -Schließen in das Programm ein. Teste Dein Programm.*

Die GUI verfügt nun über die Basisfunktionen: Man kann Bilder laden und speichern. Nun müssen nur noch die implementierten Algorithmen ausgeführt werden. Dazu werden sie zunächst dem BlueJ-Projekt mit Menü Edit -> Add Class from File dem Projekt hinzugefügt.

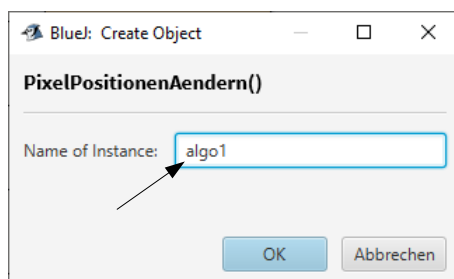
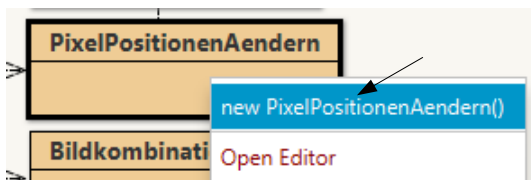
Umgang mit Klassen und Objekten

Manuell

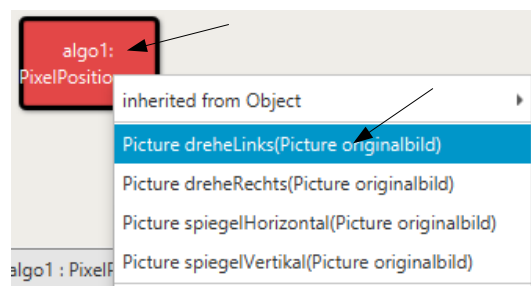
1. Attribut als Verweis erstellen

passiert implizit in Schritt 2

2. Objekt aus der Klasse erstellen



3. Methoden des Objekts aufrufen



Implementation

Deklaration eines Attributs am Anfang der Klasse:
`private PixelPositionenAendern algo1;`

In der initialize-Methode wird ein neues PixelPositionenAendern-Objekt erstellt und ein Verweis darauf in algo1 gespeichert:

```
algo1 = new PixelPositionenAendern();
```

Man kann nun Methoden des Objekts mit `objektname.methodenname(...)` aufrufen:

```
neuBild = algo1.dreheLinks(aktBild);
```




Alle Controller-Methoden, die nun noch implementiert werden müssen, sind gleich aufgebaut und verwenden die erstellten Algorithmen-Objekte. Zunächst werden aus den Controller-Elementen die notwendigen Informationen ausgelesen, dann werden Berechnungen durchgeführt und am Ende die Controller-Elemente aktualisiert.

z.B. in `mDreheLinks()` :

```
// Auslesen von Control-Elementen
Picture aktuellesBild = viewer.getImage();

// Berechnung durchführen
Picture neuesBild = algo1.dreheLinks(aktuellesBild);

// Control-Elemente anpassen
viewer.setImage(neuesBild, true); // true = altes Bild speichern
```

Genauso werden die anderen Menü-Methoden implementiert. Um zum Testen des Programms nicht immer wieder das Bild öffnen zu müssen, kann in der `initialize`-Methode automatisch ein Bild geöffnet werden:

```
Picture neuesBild = new Picture("images/katze.jpg");
viewer.setImage(neuesBild, false);
```

Der Zurück-Button ist sehr einfach zu programmieren, da der `PictureViewer` eine Methode bereitstellt (`viewer.back()`), um zum vorherigen Bild zurückzukehren. Diese muss lediglich beim `viewer` aufgerufen werden.

Der Beenden-Menüpunkt erfordert ein bisschen mehr Aufwand. Man muss ein Element nach dem Fenster fragen, in dem es angezeigt wird. Dieses Fenster kann man dann schließen.

```
Stage stage = (Stage) hauptbereich.getScene().getWindow();
stage.close();
```

11. Video: [gui_erstellen_8.mp4](#).



Implementiere alle Action-Methoden, für die die notwendigen Algorithmen schon implementiert wurden.

Slider

Der Zoom-Slider erfordert etwas mehr Arbeit, da bei Slidern keine Action-Methode im `SceneBuilder` definiert werden kann. Daher muss in der `initialize`-Methode diese Zuordnung erfolgen. Man fügt eine sogenannte Listener-Methode hinzu, die auf Änderung der Eigenschaft `Value` (= `Sliderposition`) reagiert:

```
slZoom.valueProperty()
    .addListener((observable, oldValue, newValue) -> zoom());
```

Dadurch legt man fest, dass die Methode `zoom` aufgerufen werden soll. In der `zoom`-Methode holt man sich die `Sliderposition` und ruft die `setZoom`-Methode beim `viewer` auf.

```
void zoom() {
    double zoom = slZoom.getValue();
    viewer.setZoom(zoom);
}
```

12. Implementiere die Zoom-Methode für den Slider.





Informationsdialog

Üblicherweise wird bei jedem Programm ein Info-Fenster über Autor, Version usw. im Hilfe-Menü angezeigt. Dieses Informationsfenster kann man in Java mit einer Alert-Box realisieren.

Informiere dich im Internet über die Verwendung einer Alert-Box: z.B. unter <https://code.makery.ch/blog/javafx-dialogs-official/> (Stand 08.01.2020)

Bei den Codebeispielen werden oft die notwendigen Import-Anweisungen weggelassen. Darauf deutet die Fehlermeldung "cannot find symbol" hin. Suche in diesem Fall nach "javaFx" und dem entsprechenden Bezeichner. In den Oracle Docs findet man dann eine derartige Seite:


The screenshot shows the Oracle JavaFX API documentation for the class `javafx.scene.control.Alert.AlertType`. The navigation bar at the top includes links for OVERVIEW, PACKAGE, CLASS (highlighted), USE, TREE, DEPRECATED, INDEX, and HELP. Below this, there are links for PREV CLASS, NEXT CLASS, FRAMES, NO FRAMES, and ALL CLASSES. The main content area shows the package `javafx.scene.control` and the class `Enum Alert.AlertType`. It also shows the inheritance hierarchy: `java.lang.Object` and `java.lang.Enum<Alert.AlertType>`. The class `javafx.scene.control.Alert.AlertType` is highlighted with a red box and an arrow. Below this, it lists the interfaces implemented: `Serializable` and `Comparable<Alert.AlertType>`. Finally, it shows the enclosing class: `Alert`.

Dort ist vermerkt, wo diese Klasse definiert ist. Diese importiert man dann mit der entsprechenden Import-Anweisung:

```
import javafx.scene.control.Alert.AlertType;
```

oder man importiert alles, was zu Alert gehört mit der Anweisung

```
import javafx.scene.control.Alert.*;
```

13. Implementiere einen Informationsdialog mit Informationen über dein Programm (Autor,  Version, Datum usw).