GUI PROGRAMMIERUNG

ZPG IMP KLASSE 10

HINTERGRUND

Thomas Schaller – E-Mail: thomas.schaller@zsl-rska.de. – April 2020

Inhaltsverzeichnis

MVC-Softwaremuster

Das MVC-Softwaremuster ist eine Möglichkeit zur Strukturierung einer Anwendung. Dazu werden drei Bereiche unterschieden: Datenmodell (model), Darstellung (view) und Programmsteuerung (control).

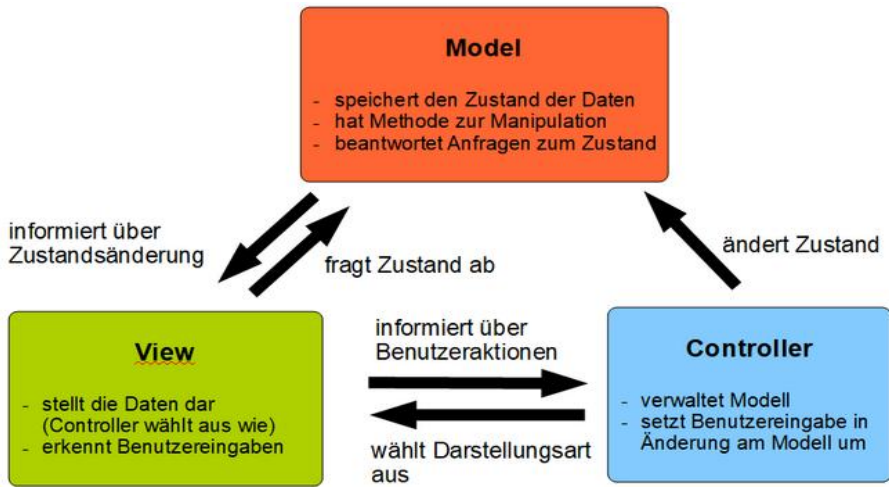


Abb. 1: MVC-Modell (eigenes Werk)

Durch diese Trennung erleichtert man die Wiederverwendung einzelner Teile der Software. Ohne Änderung des Datenmodells kann die Software auf andere Geräte oder sogar andere Betriebssysteme adaptiert werden.

Modell (model)

Das Modell enthält die darzustellenden Daten. Es ist von Präsentation und Steuerung unabhängig. Das Modell muss die Änderungen von relevanten Daten an die „Beobachter“ melden. Als Beobachter werden die View-Elemente der Anwendung eingetragen.

Präsentation (view)

Die Präsentationsschicht ist für die Darstellung der benötigten Daten aus dem Modell und die Entgegennahme von Benutzerinteraktionen zuständig. Sie kennt sowohl ihre Steuerung als auch das Modell, dessen Daten sie präsentiert, ist aber nicht für die Weiterverarbeitung der vom Benutzer übergebenen Daten zuständig. Im Regelfall wird die Präsentation über Änderungen von Daten im Modell unterrichtet und kann daraufhin die aktualisierten Daten abrufen.

Steuerung (controller)

Die Steuerung verwaltet eine oder mehrere Präsentationen, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend. Zu jeder Präsentation existiert eine Steuerung. Es ist die Aufgabe der Steuerung, Daten zu manipulieren. Die Steuerung entscheidet aufgrund der Benutzeraktion in der Präsentation, welche Daten im Modell geändert werden müssen. Sie enthält weiterhin Mechanismen, um die Benutzerinteraktionen der Präsentation einzuschränken. Die Steuerung kann in manchen Implementierungen ebenfalls zu einem „Beobachter“ des Modells werden, um bei Änderungen der Daten den View direkt zu manipulieren.[1] Die direkte Interaktion zwischen Model und View entfällt dann. Model und View sind nur über den Controller verbunden.

1 Definition des MVC-Musters nach Seite „Model View Controller“. In: Wikipedia, Die freie Enzyklopädie. (25.10. 2013). URL: http://de.wikipedia.org/w/index.php?title=Model_View_Controller&oldid=123795562

Umsetzung in Java

Die strenge Umsetzung des MVC-Musters in drei verschiedenen Klassen ist in der Schule in der Regel ein übertriebener Aufwand. Die Trennung des Datenmodells von der Steuerung und der Darstellung ist aber sinnvoll. In der hier vorgestellten Unterrichtseinheit entsteht diese Trennung nahezu zwangsläufig, auch wenn sie nicht Thema des Unterrichts ist. Das MVC-Muster sollte daher nicht im Vordergrund stehen, kann aber natürlich als Konzept vorgestellt werden.

Die Daten bestehen in dieser Unterrichtseinheit aus Bildern (Klasse Picture). Die Daten werden mit Algorithmen der Bildbearbeitung verändert. Diese Algorithmen können unabhängig von View und Controller ausgeführt und in BlueJ getestet werden. BlueJ (und die Klasse PictureViewer) stellt sozusagen automatisch einen View und den dazugehörigen Controller bereit.

Mit dem Gluon Scene Builder wird die Oberfläche (View) entworfen. Dort werden auch die (meisten) Listener-Methoden festlegt. Dadurch wird die Darstellung weitgehend von der Steuerung getrennt. Der Scene Builder bietet aber automatisch einen Rohling für die Controller-Klasse an.

Die Controller-Klasse stellt auch in diesem Projekt die Schnittstelle zwischen GUI und den Bilddaten dar. Viele Methoden sind daher sehr schlank und rufen nur die entsprechenden Algorithmen zur Bildbearbeitung auf.

Nicht sichtbar ist, dass der PictureViewer der GUI als Beobachter für das Bild eingetragen wird. Jede Änderung an einem Bild durch Aufruf einer Zeichenmethode, würde dem eingetragenen PictureViewer-Objekt mitgeteilt, das daraufhin ein Neu-Zeichnen auslöst.

Graphical User Interface (GUI) mit dem Gluon Scene Builder

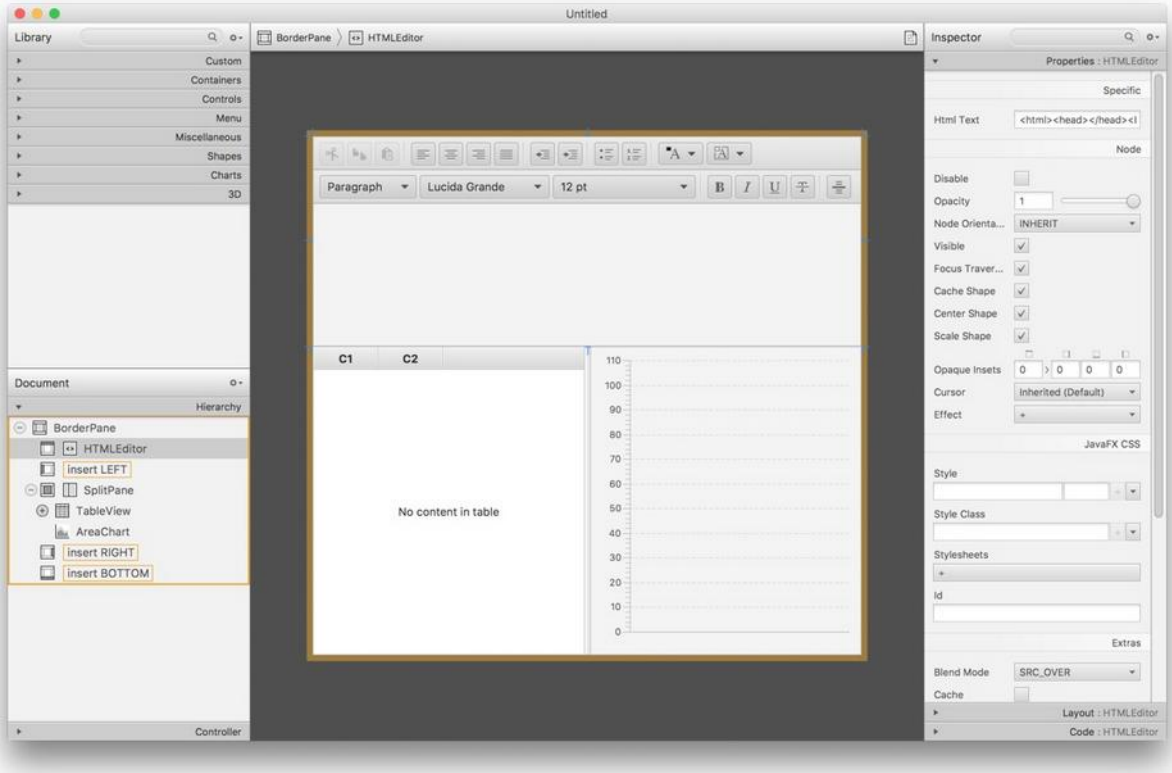
Im Bildungsplan ist verankert, dass die Schüler GUIs erstellen sollen. Dabei ist nicht daran gedacht, jedes Detail der GUI-Programmierung zu behandeln. Dies würde den Rahmen komplett sprengen. Ziel ist es, den Schülerinnen und Schülern zu vermitteln, welche Schritte notwendig sind, um ein Projekt mit einer ansprechenden GUI zu ergänzen.

Für die Erstellung einer GUI ist ein GUI-Builder hilfreich und sollte auch im Unterricht zum Einsatz kommen:

Der **JavaEditor** von G. Röhner hat einen eingebauten GUI-Builder, der die in der GUI vorgenommenen Einstellungen ständig 1:1 in Quelltext umsetzt. Änderungen im Quelltext werden aber nicht in den GUI-Builder übernommen.

Seit Java 8 gibt es die Möglichkeit in JavaFX die Definition des GUI in einer XML-Datei (fxml-Dateiendung) vorzunehmen. Der **Gluon Scene Builder**[2] bietet ein professionelles Tool, um GUIs zu erstellen und als fxml-Datei zu speichern. Diese XML-Dateien können dann mit einem einfachen JavaFX-Programm geladen werden. Dieses Programm kann den Schülern vorgegeben werden, da es immer gleich aussieht. Zusätzlich wird eine Controller-Klasse definiert, die alle Methoden beinhaltet, die bei Auswahl eines Menüpunktes oder eines Buttons ausgeführt werden sollen. Die verwendete Programmierumgebung ist dabei unabhängig von der Erstellung der GUI. Der hier vorliegende Unterrichtsgang nutzt diesen Gluon Scene Builder in Kombination mit BlueJ als Programmierumgebung.

2 Scene Builder ist ein frei verfügbares Open Source Projekt der Firma Gluon. Download des Scene Builders: *https://gluonhq.com/products/scene-builder/* (20.11.2019)

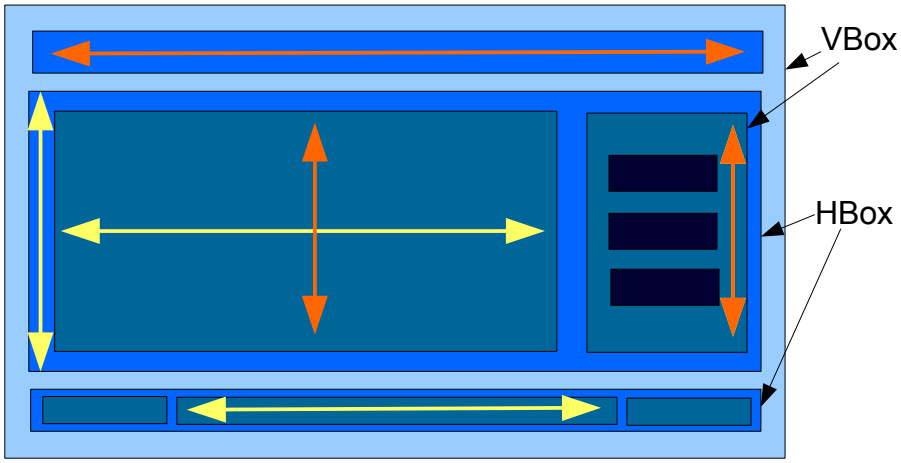


Der Scene Builder speichert das Design als FXML-Datei - also ein XML-Format für die Definition der Oberfläche. Diese Datei kann recht leicht in einem Java-Programm eingelesen und in eine Oberfläche umgewandelt werden.

```
...
     <Label text="Stricheln">
        <font>
           <Font name="System Bold" size="14.0" />
        </font>
     </Label>
     <Label text="Strichlänge:">
        <VBox.margin>
           <Insets top="10.0" />
        </VBox.margin>
     </Label>
     <Slider fx:id="sLaenge"
             majorTickUnit="5.0"
             minorTickCount="4"
             value="10.0"
             max="30.0"
             showTickLabels="true"
             showTickMarks="true"
             snapToTicks="true"/>
...
```

Der Scene Builder enthält eine Vielzahl von Einstellungsmöglichkeiten, die in der Schule nicht alle benötigt werden. Grundsätzlich besteht ein Oberflächen-Design aus einer Reihe verschachtelter Layout-Komponenten. Die wichtigsten sind VBox (ordnet die untergeordneten

Elemente vertikal an) und die HBox (ordnet die untergeordneten Elemente horizontal an). 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). Dadurch wird nur die Gliederung der Seite beschrieben und kann sich daher gut anpassen, wenn die Größe des Fensters sich ändert. Alternativ kann man mit einer AnchorPane die untergeordneten Elemente an bestimmten Positionen verankern.



Die untergeordneten Elemente können verschiedene Kontroll-Elemente sein: die wichtigsten sind Label, TextField, Button, Slider, Menus, MenuItems. Ergänzt werden die Standard-Kontrollelemente durch zwei von der ZPG IMP bereitgestellte Elemente[3]: PictureViewer (kann ein Bild anzeigen, scrollen, zoomen und den Verlauf speichern) und NumberField (man kann nur Zahlen eingeben und diese als Int oder Double abfragen).

Die Verknüpfung der Kontroll-Elemente mit dem Java-Programm erfolgt über die Festlegung der Controller-Klasse (Reiter "Controller"), der fx:id (Reiter: "Code") und der ActionListener-Methoden.

Controller: Ganz wichtig! Der Name der Controller-Klasse muss hier eingetragen werden, damit die GUI korrekt mit einer Instanz der Controller-Klasse assoziiert wird. Dieses Controller-Objekt wird normalerweise automatisch erstellt, wenn die FXML-Datei interpretiert wird.

Die fx:id (Name des Elements) ist notwendig, wenn man die Attribute dieses Elements aus dem Java-Programm heraus auslesen oder ändern möchte. Der Name ist dann der Name der Referenzvariable in der Controller-Klasse. Über die Compilerdirektive @FXML wird gekennzeichnet, dass dieses Attribut mit einem GUI-Element verknüpft werden soll.

Listener-Methoden: Bei onAction muss der Namen der Methode eingetragen werden, die aufgerufen werden soll, wenn der Button gedrückt oder der Menüpunkt ausgewählt wurde.

3 Die JAR-Datei wurde mit Java 11 erstellt. Falls man mit einer älteren Version arbeitet, muss man die JAR-Datei erneut erstellen. Dazu muss man in BlueJ in den Unterordner imp wechseln und Menü Project -> Create Jar-File wählen. Eine Main-Class muss nicht festgelegt werden.

Lambda-Ausdrücke

In Java ist es nicht möglich, Funktionen als Parameter zu übergeben. Ist dies nötig, werden Schnittstellen (Interfaces) definiert, die die notwendige Methode abstract definieren. Wollte man eine Methode übergeben, musste man vor Java 8 eine Klasse definieren, die dieses Interface implementierte und dann ein Objekt dieser Klasse übergeben. Durch die Definition des Interfaces war sichergestellt, dass das Objekt die notwendige Methode enthielt.

Beispiel: Sortieren von Arrays

```
class Vergleich implements Comparator<Integer> {
   int compare(Integer zahl1, Integer zahl2) {
   if(zahl1 > zahl2) return 1; else return 0;
}
…
Arrays.sort(zahlenliste, new Vergleich());
```

Seit Java 8 gibt es ein neues Sprachelement – den Lambda-Ausdruck, der es erlaubt, dies kompakter zu schreiben. Immer wenn eine Schnittstelle nur eine einzige Methode enthält – man nennt sie dann funktionale Schnittstelle –, dann kann ein Lambda-Ausdruck eingesetzt werden.

Beispiel: Array-Sortierung

```
Arrays.sort(zahlenliste,
         (Integer zahl1, Integer zahl2) ->
         {if(zahl1  >  zahl2) return 1; else return 0;}
);
```

Beispiel: Action-Listener

```
sLaenge.valueProperty().addListener(
         (observable, oldValue, newValue) -> stricheln(alt)
);
```

Die Methode addListener erwartet eigentlich ein Objekt einer Klasse, die die Schnittstelle ChangeListener implementiert. In dieser Schnittstelle ist eine Methode `changed(ObservableValue observable, T oldValue, T newValue)` definiert. Die Parameter dieser Methode müssen in einem Lambda-Ausdruck in der ersten Klammer stehen, dann folgt das Zeichen -> und dann die Befehle, die die abstrakte Methode changed implementieren. In diesem Befehl wird also eine Klasse definiert, die ChangeListener implementiert, wobei der Code automatisch der (einzigen) Methode zugeordnet wird, die dieses Interface enthält. Davon wird ein Objekt erzeugt und der Methode addListener übergeben.

Das klingt alles sehr kompliziert, lässt sich aber sehr intuitiv nutzen. Die SuS sollen an dieser Stelle nicht die Hintergründe von Lambda-Ausdrücken verstehen. Dieses würde fundierte Kenntnisse der objektorientierten Programmierung voraussetzen. Sie nutzen dieses Sprachmittel, indem sie den vorgegebenen Quelltext anpassen, um eigene Lambda-Ausdrücke zu erstellen.