Tipps zur Implementierung einfacher Programme mit GUI

Nachfolgend werden Beispiele zur Lösung typischer Probleme bei Erstellung erster Programme mit Grafischem User-Interface gegeben. Für die Nutzung im eigenen Programm sind eventuell kleine Anpassungen notwendig.

Strukturierung

Pakete

Es ist sinnvoll, sein Projekt in zwei Pakete "gui" und "logic" aufzuteilen. Alle JavaFX-nutzenden Klassen gehören zum GUI und somit in das Paket (s.a. **Programmaufteilung**). Verschiebt man FXML-Datei und ihren Controller in ein andersnamiges Paket, so muss die Position des Controllers in der FXML-Datei angepasst werden (funktioniert nicht über Refactoring). Hierfür kann man im SceneBuilder ganz links unten unter Controller die Controller-Klasse angeben oder direkt in der FXML-Datei das Attribut des äußersten Containers *fx:controller="gui.FXMLDocumentController"* mit dem passenden Paketnamen vor dem Punkt versehen.

Bilder ablegen

Bilddateien sollte man in einem Unterverzeichnis des src-Verzeichnisses ablegen, gerne als Unterverzeichnis des Paketes "gui". Die Bilder werden somit in die jar eingebunden. Alle Pfadangaben im Programm beziehen sich auf das src-Verzeichnis (s.a. **Programmaufteilung**).

Spielsituation komplett übernehmen

Soll ein Konstruktor oder eine Methode eine Spielsituation als String erhalten, der hinterher geparst (in andere Datenformen umgesetzt) werden muss, so ist es sinnvoll, dass hier der Parametertyp InputStream verwendet wird. So wird von der Quelle der Daten abstrahiert und der ladeneden Klasse ist es egal, ob die Daten aus einem String, einer Datei von der Festplatte, der Standardeingabe, einem HTTP-Server,... kommen.

Beispiel: Habe ich die Methode

```
public void setSituation(InputStream s) {
    Scanner sc = new Scanner(s);
    String firstLine = sc.nextLine(); //get the first line
    sc.close();
}
```

und überlade diese für einen Parameter vom Typ String und vom Typ File

```
public void setSituation(String s) {
    setSituation(new ByteArrayInputStream(s.getBytes()));
}
public void setSituation(File f) {
    setSituation(new FileInputStream(f));
}
```

so kann ich `setSituation()` sowohl mit einem String als auch mit einer Datei aufrufen:

```
setSituation("Dies ist ein Test");
setSituation(new File("test.txt"));
```

JavaFX

Erzeugen von ImageViews und Einbinden in die Zellen eines GridPanes

Ein Spielbrett enthält mehrere Zellen, die durch eine eigene Klasse dargestellt werden können. Während das Spielbrett möglichst viel von der Spiellogik abbildet, kann auch schon eine Zelle Eigenschaften besitzen und Methoden ausführen. Übergibt man der **Zelle** im Konstruktor bereits ein ImageView oder ein Pane, so kann die Zelle sich darauf darstellen. Wurde *null* übergeben oder ein anderer Konstruktor genutzt, so erfolgt einfach keine Darstellung, die restliche Funktionalität der Zelle kann aber genutzt werden. Diese Variante ist für die Bonusaufgabe ausreichend.

[Im Programmierpraktikum ist eine noch bessere Trennung von Grafik und Logik verlangt, die auch die so notwendige Prüfung auf *null* überflüssig macht. Sie besteht darin, der Zelle eine Klasse **JavaFXGUI** zu übergeben, die Methoden zur Darstellung auf dem entsprechenden ImageView oder Pane bereitstellt.]

Das **Spielbrett** (bzw. die JavaFXGUI) bekommt in seinem Konstruktor u. A. ein Array von ImageViews übergeben und kann entsprechend Zellen erzeugen, die jeweils ein ImageView erhalten. Auch hier sollte zum Testen ein weiterer Konstruktor existieren, der keine ImageViews erhält.

Im SceneBuilder legt man ein GridPane entsprechend dem Spielfeld fest und vergibt eine fx-id.

Im **FXMLDocumentController** müssen die ImageViews erzeugt werden, bevor eine Instanz des Spielbretts erzeugt wird:

```
@FXML
private GridPane gridPane;
/**
```

```
     * creates an array of imageviews corresponding to the gridPane.
     * Each imageView becomes a child of the gridPane and fills a cell.
     * For proper resizing it is binded to the gridPanes width and height.
     * @return an array of imageviews added to the gridPane
     */
    private ImageView[][] initImages() {
        int colcount = gridPane.getColumnConstraints().size();
        int rowcount = gridPane.getRowConstraints().size();
        ImageView[][] imageViews = new ImageView[colcount][rowcount];
        // bind each Imageview to a cell of the gridpane
        int cellWidth = (int) gridPane.getWidth() / colcount;
        int cellHeight = (int) gridPane.getHeight() / rowcount;
        for (int x = 0; x < colcount; x++) {
            for (int y = 0; y < rowcount; y++) {
                //creates an empty imageview
                imageViews[x][y] = new ImageView();

                //add the imageview to the cell and
                //assign the correct indicees for this imageview, so you later can use GridPane.getColumnIndex(Node)
                gridPane.add(imageViews[x][y], x, y);

                //the image shall resize when the cell resizes
                imageViews[x][y].fitWidthProperty().bind(gridPane.widthProperty().divide(colcount));
                imageViews[x][y].fitHeightProperty().bind(gridPane.heightProperty().divide(rowcount));
            }
        }
        return imageViews;
    }
```

In der Zelle (bzw. der JavaFXGUI) kann jetzt mit `imgView.setImage(new Image("/gui/img/Zelle.jpg"))` ein Bild angezeigt werden.

GridPane per Code erstellen

Möchte man ein GridPane im Code erstellen, so sollte man dem neu erzeugten Grid für die gewünschte Anzahl an Reihen **RowConstraints** und für die Spalten **ColConstraints** zufügen. Für beide Constraints sollte jeweils die **MinHeight** gesetzt werden. Soll sich das GridPane an die Größe der umgebenden Komponente anpassen, so muss es dieser Komponente zugewiesen werden (z.B. borderPane.setCenter(newGridPane)). Erst anschließend sollten den Zellen Komponenten wie oben beschrieben zugewiesen werden.

Bestimmen der Koordinate der angeklickten Zelle

In einem OnMouseClickGridPane-Ereignis muss die Koordinate der angeklickten Zelle bestimmt werden, um angemessen reagieren zu können. Wurden dem GridPane ImageViews zugefügt, so kann man anhand der Koordinaten der ImageViews prüfen, innerhalb wessen Grenzen der Mausklick liegt. Dafür muss beim Zufügen der ImageViews zum GridPane eine Zuordnung zu einer Zellenkoordinate erfolgt sein. Wird im SceneBuilder ein GridPane erstellt, so kann der Name der auszuführenden Routine beim onMouseClicked-Ereignis angegeben und Code wie folgt ergänzt werden:

```
    /**
     * reacts on clicking the gridPane.
     * @param event
     */
    @FXML
    private void onMouseClickGridPane(MouseEvent event) {
        int x = -1;
        int y = -1;
        boolean leftClicked = event.getButton() == MouseButton.PRIMARY;
        boolean rightClicked = event.getButton() == MouseButton.SECONDARY;

        //determine the imageview of the grid that contains the coordinates of the mouseclick
        //to determine the board-coordinates
        for (Node node : grdPn.getChildren()) {
          if (node instanceof ImageView) {
           if (node.getBoundsInParent().contains(event.getX(), event.getY())) {
             //to use following methods the columnIndex and rowIndex
             //must have been set when adding the imageview to the grid
             x = GridPane.getColumnIndex(node);
             y = GridPane.getRowIndex(node);
           }
          }
        }

        assert (x >= 0 && y >= 0): "dem Klick ist keine Koordinate zuzuordnen";

        if (leftClicked) {
            //react on leftclick
        } else if (rightClicked) {
            //react on rightclick
        }
    }
```

Wird das GridPane per Code erstellt, so kann diese Methode ohne die @FXML-Notation erstellt werden und direkt beim Erstellen des GridPanes als Eventhandler zugewiesen werden:

```
    gridPane.setOnMouseClicked((MouseEvent me) -> {
        onMouseClickedGridPane(me);
    });
```

Zeichnen auf dem Canvas

Um auf der Oberfläche zu zeichnen, kann man gut mit einer Pane arbeiten.

```
Pane pane = new Pane();
//probably would want Pane to layer imageview ontop of canvas
//otherwise, do your own translations
Canvas canvas = new Canvas();
ImageView image = ....;
pane.getChildren().addAll(canvas,image);
GraphicsContext gc = canvas.getGraphicsContext2D();
gc.fillRect(2,2,120,120);
//etc....
```

Beenden des Programms über das Menü

Das Programm kann beendet werden, indem die stage geschlossen wird. In einem Ereignis, wie dem Auswählen eines MenuItems, muss die zugehörige stage ermittelt werden. Dafür wird ein Knoten benötigt, der der stage zugefügt worden ist:

```
/**
 * end the game when clicking Menu "Spiel/Beenden"
 * @param event
 */
@FXML
private void onClickMnItmClose(ActionEvent event) {
    // use a known node (here the gridPane because the menuitem isn't a node)
    // and get its stage
    Stage stage = (Stage) gridPane.getScene().getWindow();
    stage.close();
}
```

Dialoge

Achtet darauf, keine Dialoge aus der Bibliothek *awt* oder von *Swing* (J...) zu verwenden, damit ein reines JavaFX-Programm entsteht (kontrolliert am Besten Eure Importe). Einen Dialog zur Information (Ok), zur Bestätigung (Ja/Nein) oder mit mehreren Auswahlmöglichkeiten erstellt man am besten mit einem **Alert** (schaut Euch auch die anderen AlertTypes an, um den passenden Dialog auszuwählen):

```
    Alert alert = new Alert(AlertType.CONFIRMATION);
    alert.setTitle("Titel des Fensters");
    alert.setHeaderText("Du bist ein toller Typ!");
    alert.setContentText("Möchtest Du deswegen eine Fanfare hören?");
    ButtonType btnYes = new ButtonType("Ja");
    ButtonType btnNo = new ButtonType("Nein");
    //dem folgenden Button die selben Eigenschaften wie dem Schließen-Button (Kreuz rechts oben) zuweisen
    //fehlt ein Button mit diesen Eigenschaften, funktioniert auch der Schließen-Button nicht
    ButtonType btnCancel = new ButtonType("Abbrechen", ButtonData.CANCEL_CLOSE);
    //dem Alert alle Buttons zufügen
    alert.getButtonTypes().setAll(btnYes, btnNo, btnCancel);
    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == btnYes) {
        // ... user chose "Ja"
    } else if (result.get() == btnNo) {
        // ... user chose "Nein"
    } else {
        // ... user chose "Abbrechen" or closed the dialog
    }
```

Kommandozeilenparameter an JavaFX-Programme

Die einfachste und für das Programmierpraktikum ausreichende Variante ist:

Legt in der Hauptklasse (die von Application erbt) eine Variable public static an und setzt diese in der Methode main(). Das anschließende launch() muss ohne Parameter erfolgen.

```
public class Main extends Application {
   public static String[] parameters;
   public void start(Stage stage) throws Exception {
     ...
   }
   public static void main(String[] args) {
     parameters = args;
     launch();
   }
}
```

Im FXMLDocumentController kann man dann auf Main.parameters zugreifen.

Animation

Die Benutztung von Thread.sleep() für die GUI ist von JavaFX nicht vorgesehen und führt daher zu unerwartetem Verhalten.

Einer Animation sind stets der Anfangs- und Endzustand bekannt und sie führt keine logischen Operationen aus, sondern stellt lediglich mehrere Zwischenzustände bis zum Erreichen der Darstellung des Endzustandes dar.

Für eine Animation bietet JavaFX einen **AnimationTimer** an. Diese Klasse besitzt die drei Methoden start(), stop() und handle(long now). Die Methode handle() wird nach Start des Timers immer wieder aufgerufen und enthält in "now" die aktuelle Zeit in Nanosekunden. Überschreibt man also handle(), so kann man hierin festlegen, wann (bzw. in welchem Abstand) die folgenden Aktionen ausgeführt werden sollen. Da hier Elemente der Oberfläche angesprochen werden sollen, müssen Referenzen auf diese in die Klasse gegeben werden (z.B. im Konstruktor). Um verschiedene (Spiel-)Zustände abbilden zu können, ist eine weitere Methode (z.B. updateGUI) notwendig, die diese Zustände als Parameter in einer Liste erhalten kann und start() aufruft. Ein Beispiel für diese Verfahrensweise gibt es **hier**.

Um die Verschiebung einer grafischen Komponente zu animieren, bietet sich als spezielle Animation eine **TranslateTransition** an. Dieser kann bei der Erstellung mitgegeben werden, wie lange sie daueren soll und welche Komponente animiert werden soll. Das Setzen von Start- und Endpunkt definiert den Weg.
Eine erstellte Transition wird mit der Methode *play()* gestartet.
Möchte man mehrere Verschiebungen nacheinander ausführen, so fügt man alle einer **SequentialTransition** zu, sollen mehrere Verschiebungen gleichzeitig erfolgen, so fügt man diese einer **ParallelTransition** zu.
Eine einfache Einführung in Animationen findet man bei **Oracle**. In **unserem** konkreten Beispiel enthält eine Klasse *JavaFXGUI* eine Methode *animateDrop()*, die in einem Grid das Löschen mehrerer Bilder in einer Spalte, Herunterfallen der verbleibenden Bilder und Hineinrutschen neuer Bilder animiert.

Fortgeschrittene Teilnehmer können auch mit **Threads** arbeiten, was für unsere Aufgaben allerdings Mehraufwand bedeutet. Für diesen Weg können wir leider keine Hilfe anbieten, weisen aber darauf hin, dass eine Nutzung von Threads ohne korrekte Dokumentation und Auswahl von Synchronisationsprimitiven (synchronized, notify, wait) automatisch unzureichend ist.

Kontakt FH Wedel

Staatlich anerkannte
Fachhochschule Wedel
Gemeinnützige Schul-
gesellschaft mbH
Feldstraße 143
D - 22880 Wedel
Tel.: +49 (0)4103 - 8048-0
Fax: +49 (0)4103 - 8048-39
E-Mail: sekretariat@fh-wedel.de