Das zugehörige Progamm FIGURES3DEXAMPLEWITHLIGHT produziert eine Ausgabe, wie sie rechts in Abbildung 5-24 gezeigt ist. Man erkennt dort die andere Beleuchtung.

5.6 Neuerungen in JavaFX 8 Update 40

JavaFX wurde mit Erscheinen von Java 8 bereits um diverse Bedienelemente und Funktionalitäten erweitert, jedoch gab es bis dato keine Dialoge oder Meldungsboxen. Dieses Manko wird mit Java 8 Update 40 adressiert. Neben Dialogen wurden auch Spinner-Controls sowie filter- und formatierbare Textfelder in JavaFX aufgenommen. Diese Erweiterungen schauen wir uns anhand kleiner Beispiele an.

5.6.1 Dialoge

JavaFX bietet nun im Package `javafx.scene.control` die Klassen `Alert`, `ChoiceDialog` sowie `TextInputDialog`, um auf einfache Weise attraktive Dialogboxen bereitstellen zu können.

Im folgenden Listing sehen wir in der `start()`-Methode den Aufruf verschiedener Methoden wie `showConfirmationDialog()` oder `showTextInputDialog()` sowie deren Implementierung.

```
@Override
public void start(final Stage stage)
{
    final Optional<ButtonType> result = showConfirmationDialog();

    if (result.isPresent() && result.get() == ButtonType.OK)
    {
        final Optional<String> enteredText = showTextInputDialog();
        System.out.println(enteredText);

        final Optional<String> selectedNickName = showNickNameSelectDialog();
        System.out.println(selectedNickName);
    }
}

private Optional<ButtonType> showConfirmationDialog()
{
    return new Alert(Alert.AlertType.CONFIRMATION,
                     "Do you want to learn more about dialogs?").showAndWait();
}

private Optional<String> showTextInputDialog()
{
    final TextInputDialog textDialog = new TextInputDialog("What's your name?");

    textDialog.setTitle("Text Input Dialog");
    textDialog.setHeaderText("This is a Text Input Dialog");
    textDialog.setContentText("Please enter your name:");
    textDialog.setGraphic(new ImageView());

    return textDialog.showAndWait();
}
```

```
private Optional<String> showNickNameSelectDialog()
{
    final ChoiceDialog<String> dialog = new ChoiceDialog<>("Iron", "Dark",
                                                           "Lord", "Dragon");

    dialog.setTitle("Nickname Selection");
    dialog.setHeaderText("");
    dialog.setContentText("Please select your desired nickname:");
    dialog.setGraphic(null);

    return dialog.showAndWait();
}
```

Listing 5.24 *Ausführbar als* **'DIALOGSEXAMPLE'**

Abbildung 5-25 zeigt die ersten zwei Dialoge aus dem Programm DIALOGSEXAMPLE.

Abbildung 5-25 *Dialoge*

Da ich hier nur einen kleinen Ausschnitt aus den Möglichkeiten der Dialogboxen zeigen kann, verweise ich Sie für weiterführende Details auf folgende Onlinequellen:

- `https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Dialog.html`
- `http://code.makery.ch/blog/javafx-dialogs-official/`
- `http://de.slideshare.net/btnrouge/javafx-dialogs`

5.6.2 Neuerungen in Bedienelementen

Wie zuvor erwähnt, wurde mit Java 8 Update 40 in JavaFX das Bedienelement `Spinner` hinzugefügt und das `TextField` um Formatierungsfunktionalität ergänzt.

Das Bedienelement `Spinner`

Das Bedienelement `Spinner` kombiniert zwei Pfeilknöpfe mit einem Textfeld und erlaubt durch die Knöpfe, den dargestellten Zahlenwert zu erhöhen bzw. zu verringern.

In JavaFX lässt sich ein `Spinner` auf viele Arten konfigurieren. Zum einen kann die Schrittweite angegeben werden. Zum anderen lassen sich sowohl `int`- als auch `double`-Werte verarbeiten. Schließlich gibt es noch verschiedene Varianten der Darstellung, die sich in der Positionierung der Pfeilknöpfe unterscheiden, wie dies in Abbildung 5-26 gezeigt ist.

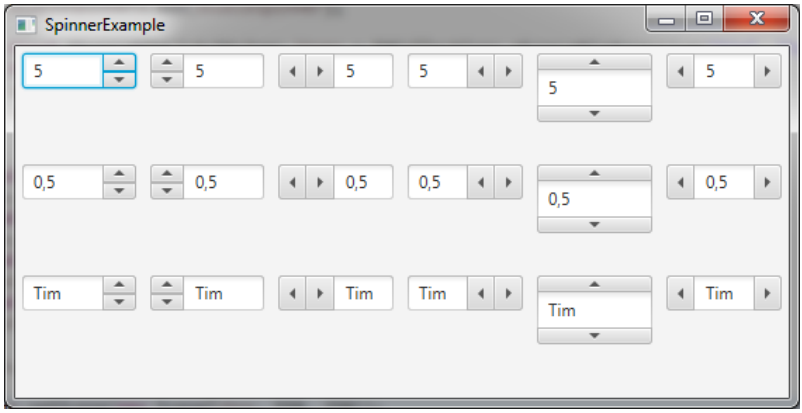

Abbildung 5-26 *JavaFXSpinnerExample*

Das nachfolgende Programm SPINNEREXAMPLE produziert die obige Darstellung und basiert auf einem Scala-Programm, das ich im Internet[11] gefunden habe und hier leicht modifiziert präsentiere:

```
@Override
public void start(final Stage stage)
{
    final String[] styles = {
                    "spinner", // Spinner.DEFAULT_STYLE_CLASS // kein Zugriff
                    Spinner.STYLE_CLASS_ARROWS_ON_LEFT_VERTICAL,
                    Spinner.STYLE_CLASS_ARROWS_ON_LEFT_HORIZONTAL,
                    Spinner.STYLE_CLASS_ARROWS_ON_RIGHT_HORIZONTAL,
                    Spinner.STYLE_CLASS_SPLIT_ARROWS_VERTICAL,
                    Spinner.STYLE_CLASS_SPLIT_ARROWS_HORIZONTAL };

    final List<Spinner<Integer>> intSpinners = new ArrayList<>();
    final List<Spinner<Double>> doubleSpinners = new ArrayList<>();
    final List<Spinner<String>> valueSpinners = new ArrayList<>();

    for (final String style : styles)
    {
       final Spinner<Integer> intSpinner = new Spinner<>(0, 100, 5);
       intSpinner.getStyleClass().add(style);

       final Spinner<Double> doubleSpinner = new Spinner<>(0.0, 1.0, 0.5, 0.01);
       doubleSpinner.getStyleClass().add(style);

       final String[] names = { "Tim", "Tom", "Jerry" };
       final ObservableList<String> items =
                                    FXCollections.observableArrayList(names);
       final Spinner<String> stringSpinner = new Spinner<>(items);
       stringSpinner.getStyleClass().add(style);

       intSpinners.add(intSpinner);
       doubleSpinners.add(doubleSpinner);
       valueSpinners.add(stringSpinner);
    }
```

[11] https://codingonthestaircase.wordpress.com/category/javafx/

```
    final HBox hbox = new HBox(10);
    hbox.getChildren().addAll(intSpinners);
    final HBox hbox2 = new HBox(10);
    hbox2.getChildren().addAll(doubleSpinners);
    final HBox hbox3 = new HBox(10);
    hbox3.getChildren().addAll(valueSpinners);

    final VBox vbox = new VBox(30);
    vbox.setPadding(new Insets(5));
    vbox.getChildren().addAll(hbox, hbox2, hbox3);

    stage.setScene(new Scene(vbox, 550, 250));
    stage.setTitle("SpinnerExample");
    stage.show();
}
```

Listing 5.25 *Ausführbar als* **'SPINNEREXAMPLE'**

Formatierung in `TextFields`

Für verschiedene Anwendungsfälle wäre es bei der Eingabe in `TextFields` wünschenswert, wenn eine spezielle Formatierung bzw. Umwandlung der Eingabe inklusive Validierung erfolgen würde, etwa um Zahlen oder Datumswerte adäquat zu verarbeiten. Mit JDK 8 Update 40 wurden diverse Konverterklassen (unter anderem `IntegerStringConverter`, `LocalDateStringConverter` und `TimeStringConverter`) sowie eine Klasse `TextFormatter` ergänzt. Mit diesen kann man die Verarbeitung in einem `TextField` steuern. Die Details habe ich nachfolgend im Listing in der Methode `createFormattedTextField()` gekapselt:

```
@Override
public void start(final Stage stage)
{
    final StringConverter<Integer> intToString = new IntegerStringConverter();
    final TextField integerTextField = createFormattedTextField(intToString);
    integerTextField.setPromptText("Bitte eine Ganzzahl eingeben!");

    final StringConverter<LocalDate> dateToString = createLocalDateConverter();
    final TextField dateTextField = createFormattedTextField(dateToString);
    dateTextField.setPromptText("Datum im Format dd.MM.yyyy eingeben!");

    final VBox vbox = new VBox(30);
    vbox.setPadding(new Insets(5));
    vbox.getChildren().addAll(integerTextField, dateTextField);

    stage.setScene(new Scene(vbox, 400, 100));
    stage.setTitle("FormattedTextFieldExample");
    stage.show();
}

private TextField createFormattedTextField(final StringConverter<?> converter)
{
    final TextField textField = new TextField();
    textField.setTextFormatter(new TextFormatter<>(converter));
    textField.setOnAction(event -> checkValidity(textField, converter));

    return textField;
}
```

```
private void checkValidity(final TextField textField,
                           final StringConverter<?> converter)
{
    try
    {
        converter.fromString(textField.getText());
    }
    catch (final RuntimeException ex)
    {
        DialogUtils.showExceptionDialog("Ungültige Eingabe", ex);
    }
}

private StringConverter<LocalDate> createLocalDateConverter()
{
    return new LocalDateStringConverter(FormatStyle.MEDIUM, Locale.GERMANY,
                                        Chronology.ofLocale(Locale.GERMANY));
}
```

Listing 5.26 *Ausführbar als* **'FORMATTEDTEXTFIELDEXAMPLE'**

Im Beispiel sehen wir im `setOnAction()`-Listener, wie man eine Validierung vornehmen kann und im Fehlerfall einen Warndialog anzeigt. Das ist Aufgabe der Methode `showExceptionDialog(String, Exception)`, die in einer eigenen Utility-Klasse `DialogUtils` basierend auf den zuvor schon beschriebenen Dialogfunktionalitäten wie folgt implementiert wird:

```
public class DialogUtils
{
    public static void showExceptionDialog(final String hint,
                                           final Exception ex)
    {
        final Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Internal Software Error");
        alert.setHeaderText(hint);
        alert.setContentText(ex.toString());

        final Pane detailsPane = createStacktracePane(ex);
        alert.getDialogPane().setExpandableContent(detailsPane);

        alert.showAndWait();
    }

    private static Pane createStacktracePane(final Exception ex)
    {
        final StringWriter sw = new StringWriter();
        final PrintWriter pw = new PrintWriter(sw);
        ex.printStackTrace(pw);

        final Label details = new Label("Stacktrace:");
        final TextArea textArea = new TextArea(sw.toString());
        textArea.setEditable(false);
        textArea.setWrapText(true);

        final FlowPane contentPane = new FlowPane();
        contentPane.getChildren().addAll(details, textArea);
        return contentPane;
    }
}
```

Starten wir das obige Programm FORMATTEDTEXTFIELDEXAMPLE und geben dann ein Datum im amerikanischen Format oder aber einen beliebigen Text ein, also irgendeine Information, die vom erwarteten und spezifizierten deutschen Format abweicht, so erhält man eine Darstellung und Fehlermeldung ähnlich wie in Abbildung 5-27.

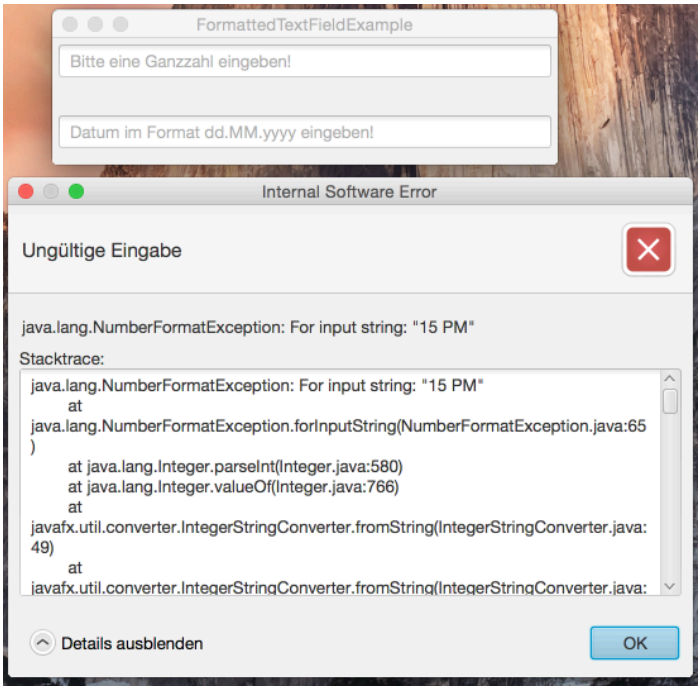

Abbildung 5-27 *FormattedTextFieldExample*

5.7 Fazit

Mit den Beispielen zu den Neuerungen aus JavaFX 8 Update 40 endet unser Überblick über JavaFX. Sie sollten mittlerweile einen guten Fundus an Wissen haben, um mit eigenen Experimenten zu beginnen. Dabei hilft das recht intuitive JavaFX-API. Damit kommt man schnell zu ersten Erfolgen, die sich dann leicht ausbauen lassen.

Abschließend möchte ich noch auf einen Punkt hinweisen: Zwar ist JavaFX ein wirklich umfangreiches GUI-Framework, aber es ist im Gegensatz zu den Werbeversprechungen von Oracle lediglich ein Rich-Client-Framework, aber keine Rich-Client-Plattform (RCP). Letztere definiert sich darüber, dass dort diverse allgemeine Basisfunktionalitäten, die für verschiedenste Applikationen in ähnlicher Form erforderlich sind, bereitgestellt werden. Auf diese Weise vermeidet man einiges an Aufwand bei der Applikationsentwicklung durch die Nutzung der Funktionalitäten aus der RCP. Ein interessanter Artikel, der das Thema im Detail behandelt, ist im Java Magazin 1.14 unter dem Titel »Wo ist die Rich-Client-Plattform für JavaFX?« erschienen.