Technische Hochschule Ingolstadt

Vorführaufgabe 10: Die GUI

Ziel dieser Vorführaufgabe ist die Implementierung eines Bediendialogs (grafische Benutzeroberfläche, engl. *graphical user interface*, kurz GUI), der die komfortable Steuerung der bisher entwickelten Funktionalität ermöglicht.

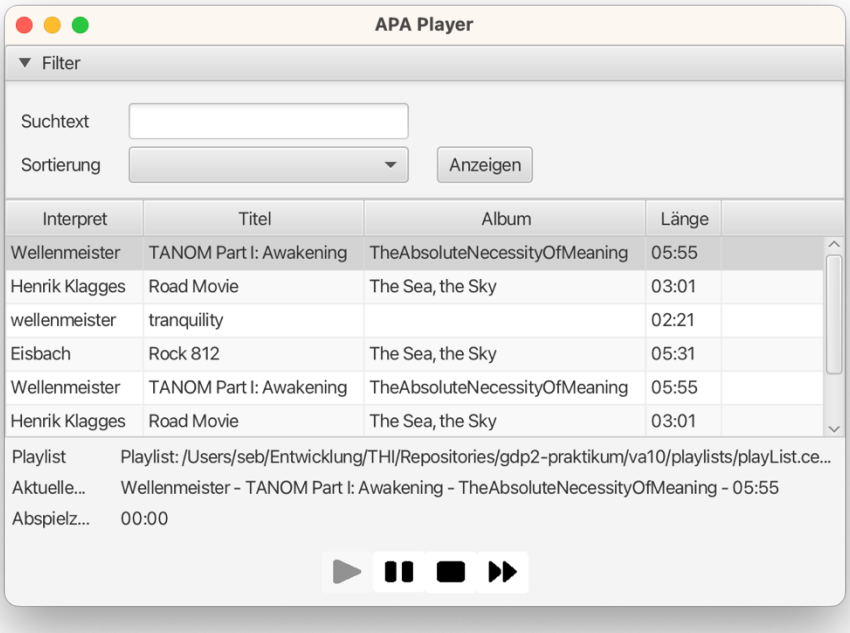


Abbildung 1: Benutzeroberfläche des Media Players

Vorbereitungen

Kopieren Sie ihr Projekt zur vorangegangenen Vorführaufgabe, entfernen Sie die Tests im cert-Ordner und ersetzen Sie die Tests im „tests“-Ordner durch die der aktuellen Vorführaufgabe (Archiv tests.zip). Die Klassen in tests.zip sind bereits in einer Package-Struktur, achten Sie auf die korrekte Ordnerstruktur im Ordner tests um das Package `studiplayer.test` abzubilden.

Es gibt weitere Archive, die sie bitte nach dem Download folgendermaßen behandeln

- „examples.zip: Enthält die einführende Anwendung „GuessANumber“
  Alle Dateien nach dem Entpacken in dem neu anzulegenden Source-Folder „examples“ speichern
- „playlists.zip“: Enthält vorbereitete Abspiellisten im M3U-Format
  Alle Dateien nach dem Entpacken im Ordner „playlists“ speichern
- „icons.zip“: Enthält Grafikdateien für die Buttons
  Alle Dateien nach dem Entpacken unter „src“ als Folder „icons“ speichern

JavaFX mit Eclipse nutzen

Bevor mit der Entwicklung einer JavaFX-Anwendung begonnen werden kann, müssen noch einige Vorbereitungen getroffen werden[1]:

1. Java-Version: Stellen Sie sicher, dass Sie Java 13 oder höher in Eclipse nutzen, siehe Abbildung unten.

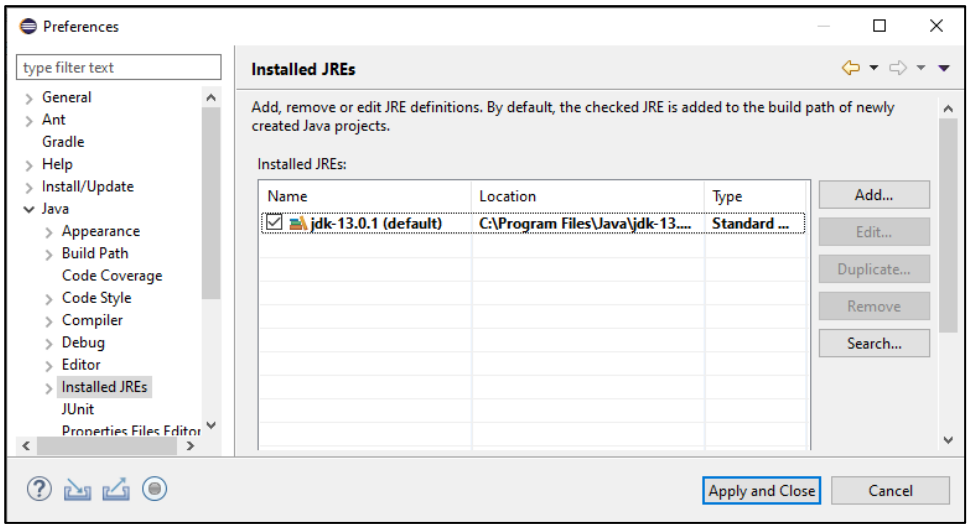


Abbildung 2 Verifizieren der eingestellten Java-Version

2. Installieren Sie das SDK (software development kit) für JavaFX von https://gluonhq.com/products/javafx/, wobei die für ihr Betriebssystem und Prozessor-Architektur passende Version auszuwählen ist. Für Windows kann die Installation z.B. unter C:\bin\javafx-sdk-11.0.2 erfolgen.
3. Legen Sie eine neue „user library“ namens “JavaFX” unter Window > Preferences > Java > Build Path > User Libraries an, indem Sie den New-Button anklicken und im nachfolgenden Dialog den Namen der Libary eingeben. Fügen Sie anschließend mithilfe des Buttons „Add external jars“ alle jar-Dateien im lib-Verzeichnis ihrer JavaFX-Installation hinzu. Für den Installationsort von Punkt 2. (Windows!) wählt man alle Dateien in C:\bin\javafx-sdk-11.0.2\lib:

[1] Siehe auch https://openjfx.io/openjfx-docs/#IDE-Eclipse.

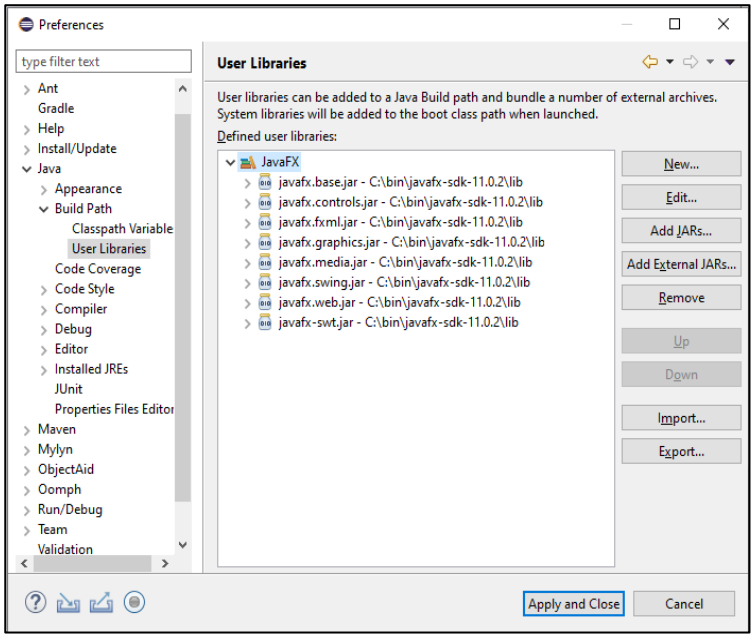


Abbildung 3: Definieren der JavaFX User Library

4. Fügen Sie die UserLibrary ihrem Build Path hinzu. Hierfür öffnen Sie den Dialog (rechte Maustaste auf dem Projektnamen im Package Explorer > Java Build Path) und fügen die neue User Library „JavaFX“ hinzu (Add Library > User Library auswählen > next > JavaFX auswählen).
5. Starten eines Programms: Mit Hilfe der User Library „JavaFX“ können ihre JavaFX nutzenden Klassen kompiliert werden. Zur Ausführung muss der JVM mitgeteilt werden, wo die JavaFX-jar’s liegen. Hierfür muss für jede Anwendung in Eclipse eine „run configuration“ (für das Debuggen analog eine „debug configuration“) erstellt werden. Klicken Sie hierfür im Package Explorer mit der rechten Maustaste auf ihre main-Klasse (z.B. Raten.java) und wählen den Menüpunkt Run As > Run Configurations aus. In dem folgenden Dialog wählen Sie „Java Application“ aus und klicken auf den New-Button (links oben im Dialog), woraufhin sich Felder zur Bearbeitung der Eigenschaften ihrer Run Configuration öffnen. Hier ist vor allem darauf zu achten, dass die Angabe

   ```
   --module-path "C:\bin\javafx-sdk-11.0.2\lib" --add-modules javafx.controls
   ```

   im Feld „VM arguments“ vorgenommen wird. Setzen Sie auch die Option “-XstartOnFirstThread”.

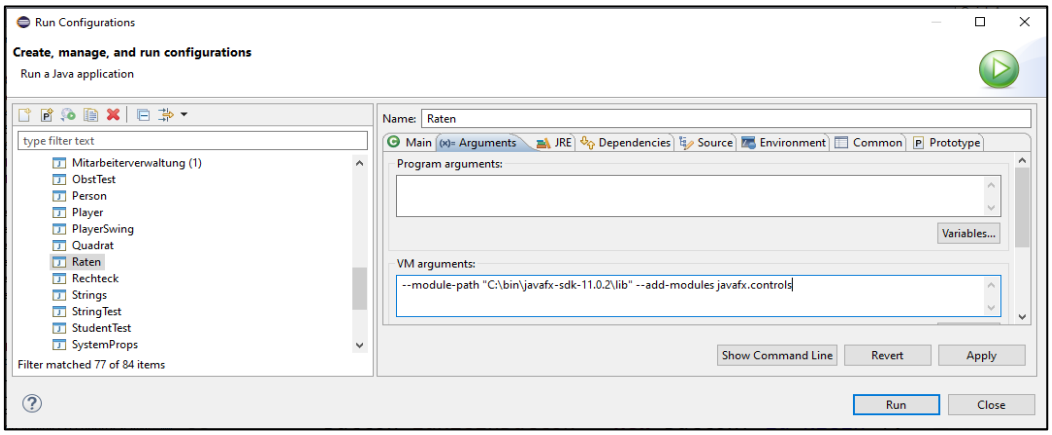


Abbildung 3 Anlegen einer Run Configuration für Raten

6. Nun sollte die Anwendung (z.B. GuessANumber) ausgeführt werden können. Nach einmaliger Ausführung kann die Anwendung auch wie gewohnt über den grünen Pfeil-Button oder über den Käfer-Button (zum Debuggen) ausgeführt werden.

Zahlenraten: ein einführendes Beispiel

Bevor Sie mit der Implementierung der grafischen Oberfläche für Ihren Media-Player beginnen, studieren Sie bitte zuvor das Beispiel GuessANumber.java im Unterverzeichnis examples.

Dieser Beispiel-Code erzeugt eine einfache grafische Oberfläche für ein Zahlenratespiel. Experimentieren Sie mit dem Code und ändern Sie

- Layout
- Event-Erzeugung
- Event-Behandlung

Hinweis: in der Klasse GuessANumber.java werden die Elemente der grafischen Oberfläche (Buttons etc.) zusammen mit der Behandlung der generierten Events (Lambda Expressions) implementiert. Dieses Vorgehen ist nur für kleine Beispiele ratsam. Bei aufwendigeren Oberflächen sollte die Behandlung der Events unbedingt in separaten Klassen erfolgen. Siehe hierzu auch OO-Entwurfsmuster MVC: (Model-View-Control).

Teilaufgabe a) Paket studiplayer.ui

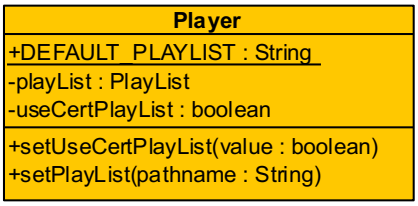
Erzeugen Sie ein neues Paket „studiplayer.ui“ und speichern Sie die vorgegebenen Java-Klassen „Song.java“ und „SongTable.java“ (siehe APA-Kursraum/VA10) dort ab.

Legen Sie nun die Klasse `Player` an. Der Player sollte folgenden initialen Anforderungen genügen:

- Player spezialisiert `javafx.application.Application` und bietet einen Default-Konstruktor
- Player implementiert eine Methode `start(Stage stage)` sowie eine `main`-Methode mit dem üblichen Inhalt zum Start der Anwendung
- Die `start()`-Methode setzt den Titel und setzt ihre Haupt-Pane mit einer Größe von 600x400 Pixeln „in Scene“.

Nach Umsetzung dieser Vorgaben sollten Sie ein leeres Fenster mit dem passenden Titel sehen.

Zum Testen der oben beschriebenen Struktur können Sie die bereitgestellten Tests in der **Datei TestSubtaskA.java** nutzen. Erstellen Sie hierfür eine „JUNIT Run Configuration“ mit einer entsprechenden Belegung des Feldes „VM arguments“.



Teilaufgabe b) Initialisieren und Laden einer Playliste

Beim Starten der Anwendung soll eine PlayList initialisiert werden. Entweder wird dem Nutzer ein Dialog präsentiert, um eine m3u-Datei auszuwählen, oder die Anwendung verwendet eine Default-m3u-Datei. Die jeweilige m3u-Datei befüllt initial die PlayList. Setzen Sie folgende Logik um:

- Für den Dialog zur m3u-Dateiauswahl durch den Nutzer soll in JavaFX der FileChooser eingesetzt werden, siehe z.B. https://docs.oracle.com/javafx/2/ui_controls/file-chooser.htm
- Für die Ausführung der Tests soll der FileChooser deaktiviert werden können, definieren Sie

ein bool'sches Attribut `useCertPlayList` und einen Setter dafür. Das Attribut sollte initial auf „false“ gesetzt werden.

> **Hinweis:** `useCertPlayList` steuert, ob zu Beginn die Default-m3u-Datei „playList.cert.m3u“ geladen wird oder der Benutzer per Dateiauswahl-Dialog eine m3u-Datei wählt, welche dann geladen wird. Diese Methode wird im Abnahmetest genutzt, um den Dialog ohne Dateiauswahl auszuführen.

Implementieren Sie zudem die Methode `loadPlayList(String pathname)`, welche die PlayList des Players anhand des übergebenen Pfadnamens neu lädt. Ist der Name „null“ oder ein Leerstring, so soll die Default-PlayList geladen werden. Vereinbaren Sie hierfür eine öffentliche Konstante namens `DEFAULT_PLAYLIST` vom Typ String mit der entsprechenden Default-PlayList „playList.cert.m3u“.

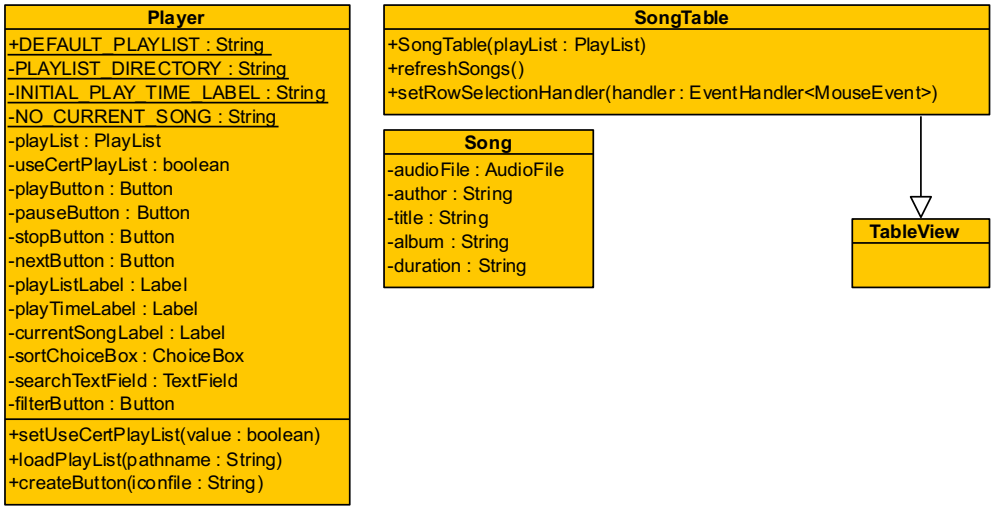
Zum Testen der oben beschriebenen Struktur können Sie die bereitgestellten Tests in der **Datei TestSubtaskB.java** nutzen.

Teilaufgabe c) Grundlayout

In dieser Teilaufgabe geht es darum, den Dialog in Bezug auf die Controls und das Layout zu entwickeln, jedoch noch ohne die Ereignisbehandlung. Erstellen Sie alle Controls wie in der Abbildung dargestellt und verwenden Sie geeignete JavaFX Panes, um das geforderte Layout zu erreichen.

> **Hinweis:** Skizzieren Sie zuerst eine Layout-Hierarchie auf einem Notizblatt, um sich die Struktur für Controls und Panes zu überlegen. Details zu Layout-Hierarchien finden Sie im Skript. Die Skizze muss nicht abgegeben werden.

Einige Controls müssen in entsprechenden Attributen referenziert werden, damit Tests korrekt funktionieren. Definieren Sie die Attribute gemäß dem folgenden Klassendiagramm.



Für das Layout können folgende Überlegungen berücksichtigt werden:

- Die PlayList-Tabelle in der Mitte des Dialogs soll bei Vergrößerung des Fensters entsprechend mitwachsen, daher bietet sich eine `BorderPane` mit einer `TableView` im center-Bereich an. Sie können hierfür die `SongTable` verwenden (siehe Klassendiagramm),

bitte beachten Sie hierzu ergänzend Hinweis 2 weiter unten.

- Im top-Bereich der `BorderPane` wird das Filterformular platziert. Hierfür bietet sich eine `TitledPane` an, siehe hierfür https://docs.oracle.com/javafx/2/ui_controls/accordion-titledpane.htm.
- Der bottom-Bereich besitzt zwei untereinander dargestellte Elemente, weshalb sich eine `VBox` als übergeordnete Pane anbietet:
  - Der erste Bereich bietet Informationen zum aktuellen Song in tabellarischer Form (vgl. Grid) an
  - Der zweite Bereich zeigt die Buttons zur Liedsteuerung an. Zur Erzeugung der Buttons kann die Methode `createButton(String iconfile)` genutzt werden, siehe Hinweis 3 unten. Der Parameter `iconfile` gibt den Dateinamen der zu ladenden Grafik-Datei an (z.B. „play.jpg“)

Erzeugen Sie entsprechende Panes und Controls und verbinden Sie die Elemente, um das in Abbildung 1 gezeigte Layout zu erreichen.

Hinweise:

1. Richten Sie folgende zusätzliche konstante Player-Attribute ein:
   - `PLAYLIST_DIRECTORY` (private): Verzeichnis, in dem die Dateiauswahl stattfindet
   - `INITIAL_PLAY_TIME_LABEL` (private): „00:00“
   - `NO_CURRENT_SONG` (private): " - " // wenn noch kein Lied ausgewählt wurde
2. Zur Erleichterung des Umgangs mit Tabellen steht die Klasse `SongTable` bereit. Eine `SongTable`-Instanz ist eine `TableView`-Instanz, die die bei ihrer Erzeugung in Form einer `PlayList` übergebenen `AudioFiles` anzeigt. Als „row model“ für jede Tabellenzeile wird die Klasse `Song` genutzt, welche die anzuzeigenden Werte (Interpret, Titel, Album, Länge) sowie eine Referenz auf das zugrundeliegende `AudioFile` enthält. Letztere ist für die Ereignisbehandlung bei Klick auf eine Zeile relevant. Der Klick-Handler wird über die Methode `setRowSelectionHandler()` gesetzt. Eine `SongTable` kann mit der Methode `refreshSongs()` aktualisiert werden, z.B. wenn eine neue m3u-Datei geladen wird.
3. Folgende Methode kann zur Erzeugung eines Buttons mit einem Icon genutzt werden:

```
private Button createButton(String iconfile) {
    Button button = null;
    try {
        URL url = getClass().getResource("/icons/" + iconfile);
        Image icon = new Image(url.toString());
        ImageView imageView = new ImageView(icon);
        imageView.setFitHeight(20);
        imageView.setFitWidth(20);
        button = new Button("", imageView);
        button.setContentDisplay(ContentDisplay.GRAPHIC_ONLY);
        button.setStyle("-fx-background-color: #fff;");
    } catch (Exception e) {
        System.out.println("Image " + "icons/"
            + iconfile + " not found!");
        System.exit(-1);
    }
    return button;
}
```

4. Korrekte Bezeichnungen für die Auswahl der Sortiermethode
   `SortCriterion`-Instanzen werden angezeigt, um die Sortiermethode auszuwählen.

Standardmäßig geben diese Elemente das Ergebnis von `name()` zurück, d. h. Großbuchstaben-Strings aus dem Aufzählungstyp. Eine einfache Möglichkeit, die in den Tabellenspalten angezeigten Begriffe zu verwenden, besteht darin, im Aufzählungstyp eine `toString()`-Methode anzugeben, die für jeden `ordinal()`-Wert einen entsprechenden String zurückgibt. Dies kann durch Definition eines Arrays mit den Namen und die Verwendung von `ordinal()` als Index beim Zugriff auf das Array erfolgen.

Zum Testen ihres Codes können Sie die bereitgestellten Tests in der Datei **TestSubtaskC.java** nutzen.

Teilaufgabe d) Ereignisbehandlung – Filtern und Sortieren der PlayList

Diese Teilaufgabe soll die Sortierfunktion der Oberfläche mit der `PlayList` verknüpfen. Nach Klick auf den „anzeigen“-Button im Filter sind die aktuell eingestellten Werte für den Suchtext sowie die Sortierung in der PlayList zu setzen, wodurch sich eine entsprechend gefilterte bzw. sortierte Abspielliste ergibt. Die SongTable ist abschließend mittels der Methode `refreshSongs()` zu aktualisieren (siehe Abbildung 4).

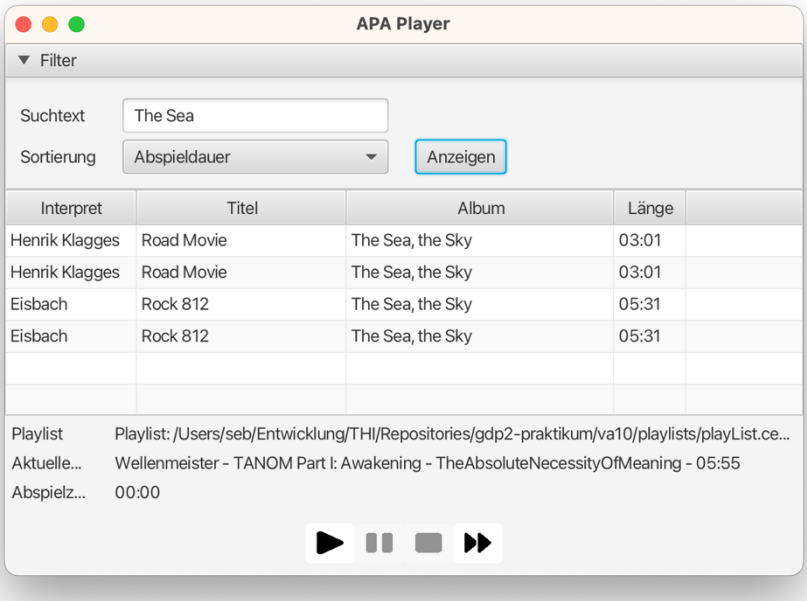


Abbildung 4: Beispiel für eine gefilterte und sortiere PlayList

Teilaufgabe e) Ereignisbehandlung – Steuerung der Abspielfolge

Die eigentliche Steuerung erfolgt über die Buttons sowie über Mausklick auf Zeilen der Tabelle („springe zu diesem Lied und spiele es ab“).

Für die Ereignisbehandlung der Buttons wird empfohlen, jeweils eine lokale Methode zu definieren und diese in dem Handler-Lambda aufzurufen. Man erhält eine besser modularisierte Anwendung mit der Möglichkeit, Code wiederzuverwenden.

Beispiel für den play-Button:

```
playButton.setOnAction(e -> {
    playCurrentSong();
});
```

In einem ersten Schritt soll nun die grundlegende Ereignisbehandlung etabliert werden. Das eigentliche Abspielen von Songs sowie das Hochzählen der Abspielzeit wird in Teilaufgabe f) programmiert. Beide Aktivitäten müssen parallel zur Benutzeroberfläche in eigenen Threads ausgeführt werden, da sonst die Benutzeroberfläche „einfrieren“ würde.

Setzen sie obige Idee für alle Buttons mit folgender initialer Ereignisbehandlung um, wobei anstelle der tatsächlichen `AudioFile`-Aktion (Abspielen/Pausieren/Stoppen) lediglich Informationen auf die Konsole auszugeben sind (siehe nachfolgende Beispiel-Ausgabe):

- Play-Button:
  → aktualisiere die Informationen zum aktuellen Song und setze die Abspielzeit auf „00:00“
  → passe die Button-Zustände (`setDisable(true|false)`) an, Abbildung 1 und Abbildung 4 zeigen den Zustand der Buttons wenn ein Lied abgespielt wird bzw. der Player gestoppt ist.
  → beginne mit der Wiedergabe
- Pause-Button:
  → falls Song abgespielt wird: pausiere die Wiedergabe
  → falls Song pausiert ist: setze die Wiedergabe fort
- Stop-Button:
  → unterbreche die aktuelle Wiedergabe
  → passe die Button-Zustände an
  → setze die Abspielzeit zurück.
- Next-Button:
  → unterbreche die aktuelle Wiedergabe, falls aktiv
  → springe zum nächsten Song
  → zeige die Song-Information an und setze die Abspielzeit auf „00:00“
  → passe die Button-Zustände an

Entsprechende Konsolausgaben für eine Beispiel-Klicksequenz

```
play > stop > play > pause > next > stop
```

könnten auf Basis der PlayList „playList.cert.m3u“ so aussehen:

```
Playing Wellenmeister - TANOM Part I: Awakening - TheAbsoluteNecessityOfMeaning - 05:55
Filename is wellenmeister_awakening.ogg

Stopping Wellenmeister - TANOM Part I: Awakening - TheAbsoluteNecessityOfMeaning - 05:55
Filename is wellenmeister_awakening.ogg

Playing Wellenmeister - TANOM Part I: Awakening - TheAbsoluteNecessityOfMeaning - 05:55
Filename is wellenmeister_awakening.ogg

Pausing Wellenmeister - TANOM Part I: Awakening - TheAbsoluteNecessityOfMeaning - 05:55
Filename is wellenmeister_awakening.ogg

Switching to next audio file: stopped = false, paused = true
Stopping Wellenmeister - TANOM Part I: Awakening - TheAbsoluteNecessityOfMeaning - 05:55
Filename is wellenmeister_awakening.ogg
Playing Henrik Klagges - Road Movie - The Sea, the Sky - 03:01
Filename is special.oGg

Stopping Henrik Klagges - Road Movie - The Sea, the Sky - 03:01
Filename is special.oGg
```

Verwenden Sie abschließend die Methode `setRowSelectionHandler` an Ihrer `SongTable`-Instanz zur Ereignisbehandlung, um ausgewählte Zeilen in der Tabelle abzuspielen. Verwenden Sie die entsprechende Logik in der `PlayList`, Stoppen Sie ggf. ein aktuell abgespieltes Lied und starten Sie das ausgewählte. Die Auswahl erhalten Sie über die Methode `getSelectionModel()`.

Hinweise:

1. Es macht Sinn, die Button-Zustände (disabled ja/nein) mit einer Hilfsmethode
   ```
   private void setButtonStates(boolean playButtonState,
       boolean pauseButtonState, boolean stopButtonState,
       boolean nextButtonState)
   ```
   zu setzen, da dies an verschiedenen Stellen benötigt wird.
2. Auch für das Aktualisieren des aktuellen Songs sowie der aktuellen Abspielzeit bietet sich an, eine Methode vorzusehen.
   Nachfolgend ist ein Grundgerüst als Startpunkt gegeben:
   ```
   private void updateSongInfo(AudioFile af) {
       Platform.runLater(() -> {
           if (af == null) {
               // hier currentSongLabel und playTimeLabel belegen
           } else {
               // hier currentSongLabel und playTimeLabel belegen
           }
       });
   }
   ```
   `Platform.runLater(<Lambda>)` bewirkt, dass `<Lambda>` in einem JavaFX-Thread parallel ausgeführt wird. Dies ist dann erforderlich, wenn z.B. der Timer-Thread (siehe Teilaufgabe f) die Abspielzeit aktualisiert. Nicht-JavaFX-Threads können nicht auf JavaFX-Controls zugreifen, dies führt zu einer Exception.
   Anmerkung: obige Methode aktualisiert stets beide Labels, was nicht immer nötig ist. Selbstverständlich können Sie eigene Methoden definieren, die dies verbessern!

Teilaufgabe f) Player- und Timer-Thread

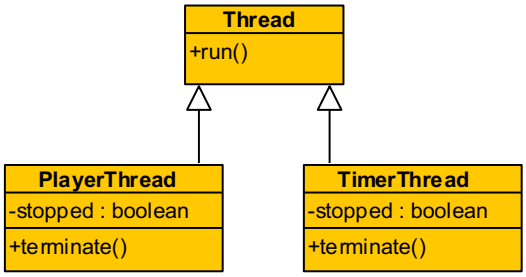
In einem letzten Schritt soll die Wiedergabe von Songs umgesetzt werden. Der naheliegende Ansatz wäre hier, in der Handler-Methode `playCurrentSong()` direkt `audioFile.play()` aufzurufen. Dies würde allerdings den Eventhandler-Thread von JavaFX blockieren, solange der Song wiedergegeben wird. Die Folge wäre, dass keinerlei Ereignisbehandlung mehr durchgeführt werden könnte, unser Dialog wäre „eingefroren“.

Die Lösung besteht darin, das Abspielen von Songs in einem eigenen Thread – den `PlayerThread` – auszuführen. Dieser soll folgende Logik umsetzen:

- Der Thread kann von außen durch Aufruf der Methode `terminate()` gestoppt werden. Diese setzt ein bool'sches Attribut `stopped`, was bewirken soll, dass die Schleife in `run()` abgebrochen wird.
- Solange der Thread nicht gestoppt wurde, wird der aktueller Song abgespielt. Vor dem Start des Abspielvorgangs wird der Song in der Tabelle selektiert (siehe Klasse `SongTable`).
- Nach der vollständigen Wiedergabe eines Songs wird zum nächsten Song in der PlayList gewechselt, falls der Thread nicht gestoppt werden soll.

Implementieren Sie den oben beschriebenen Thread z.B. als innere Klasse der Player-Klasse. Dies hat den Vorteil, dass auf Attribute (z.B. `playList`) und Methoden (z.B. `updateSongInfo()`) der Player-Klasse zugegriffen werden kann.

Starten und Stoppen Sie den `PlayerThread` im Rahmen der Ereignisbehandlung an den entsprechenden Stellen. Es bietet sich an, ein Attribut als Referenz auf die Thread-Instanz einzuführen. Am Wert ungleich null erkennt man, ob gerade ein Abspielvorgang aktiv ist oder nicht.



Für die Aktualisierung der Abspielzeit wird ebenfalls ein Thread – der `TimerThread` – eingeführt. Dieser setzt folgende Logik um:

- Analog zum `PlayerThread` kann der Timer von außen durch `terminate()` gestoppt werden.
- Solange der Thread nicht gestoppt wurde, aktualisiert er die Songinformation (Abspielzeit und aktueller Song)
- Um die Aktualisierung nicht unnötig oft durchzuführen, legt sich der Thread „schlafen“

Implementieren Sie den oben beschriebenen Thread ebenfalls als (z.B.) innere Klasse der Player-Klasse. Starten und Stoppen Sie den `TimerThread` im Rahmen der Ereignisbehandlung an den entsprechenden Stellen. Es bietet sich an, ein Attribut als Referenz auf die Thread-Instanz einzuführen, sowie ggf. Methoden, um das Starten und Stoppen der Threads zu vereinfachen.

Hinweise:

1. Die beiden Threads müssen nicht immer gemeinsam erzeugt und terminiert werden. Bei „pause“ kann der PlayerThread beispielsweise aktiv bleiben, während der TimerThread beendet und ggf. später neu erzeugt werden muss.
2. Ein modularerer Ansatz zur Umsetzung obiger Aussage wäre die Einführung und Nutzung folgender Methoden:

```
private void startThreads(boolean onlyTimer) {
    // erzeuge und starte TimerThread
    // falls !onlyTimer: erzeuge und starte PlayerThread
```

```
}
private void terminateThreads(boolean onlyTimer) {
    // terminiere TimerThread, setze thread-Attribut auf null
    // falls !onlyTimer: terminiere PlayerThread,
    // setze thread-Attribut auf null
}
```

Achtung: Testen Sie ihre Anwendung interaktiv ausgiebig, bevor Sie mit den Abnahmetests beginnen!

Hinweise zur Abnahme Ihrer Implementierung der Vorführaufgabe 10

Laden Sie alle zum Aufgabenblatt gehörigen Abnahme-Tests herunter und speichern Sie diese im Source-Folder `cert` Ihres Projekts. Dann führen Sie die Abnahme-Tests in Eclipse aus.

Hinweis: Sie können alle Tests zusammen ausführen, indem Sie die rechte Maustaste auf dem cert-Source Folder nutzen und „run as > JUNIT test“ wählen. Dies setzt jedoch voraus, dass Sie vorher analog zur direkten Ausführung des Players eine „run configuration“ angelegt haben. Auch hier ist das Feld „vm arguments“ entsprechend zu belegen!

Wenn alle Tests ohne Fehler durchlaufen, mailen Sie bitte folgende Java-Dateien als Dateianhang mit dem Betreff „VA10“ an den APA-Server:

- aus Paket `studiplayer.audio`:
  AlbumComparator.java, AudioFileFactory.java, AudioFile.java, AuthorComparator.java, ControllablePlayListIterator.java, DurationComparator.java, NotPlayableException.java, PlayList.java, SampledFile.java, SortCriterion.java, TaggedFile.java, TitleComparator.java, WavFile.java
- aus Paket `studiplayer.ui`:
  Player.java, SongTable.java, Song.java