

Abschlussprüfung Sommer 2021

Auszubildender

Luis San Martin Martinez

Fachinformatiker für Anwendungsentwicklung

# “Benutzeroberfläche für Spracherkennung mit JavaFX”

Ausbildungsbetrieb:

aicas GmbH

Emmy-Noether-Straße 9,

76131 Karlsruhe

## Abstract

Diese Dokumentation zeigt eine in JavaFX entwickelte grafische Oberfläche, die auf der Spracherkennung basiert, die von der externen CMUSphinx-Bibliothek bereitgestellt wird.

Für die Bedienung und Erkennung wesentlicher Elemente wie vorprogrammierter Sprachbefehle, die das Interaktionssystem zwischen Benutzer und Programm erleichtern, muss zunächst die von CMUSphinx bereitgestellte Entwicklungsumgebung konfiguriert werden.

Hierzu werden die von der Bibliothek bereitgestellten Sprachressourcen verwendet oder eigene Ressourcen erstellt, z. B. unabhängige grammatikalische oder sprachliche Dateien für jeden Anwendungsfall.

Tests zur Messung der Leistung während der Entwicklung der grafischen Benutzeroberfläche für das Spracherkennungssystem werden selektiv durchgeführt, jedoch nicht in allen Entwicklungsstadien. Diese laufen in zwei ähnlichen Umgebungen, jedoch mit unterschiedlichen Geräuschpegelbereichen, die dem aktuellen technischen Status der verwendeten Hardware entsprechen.

*Umgebung1: Lautsprecher*

*Umgebung2: Head-Set-Mikrofon*

Es wird hervorgehoben, dass die Qualität der erhaltenen Spracherkennung von der Aussprache und der Qualität des Audios abhängt, um schließlich die geeigneten Ergebnisse bei der Durchführung der Demonstration zu erzielen.

*Diese Parameter können jedoch durch das Training des Spracherkennungsmodells verbessert werden.*

# Inhaltsverzeichnis

## 1. Einleitung (Einführung)

### 1.1 Unternehmensprofil Auftraggeber

### 1.2 Mitwirkende Abteilungen

## 2 Zielsetzung der Projektarbeit

### 2.1 Analyse der Ausgangssituation (Umfang)

### 2.2 Lastenheft

## 3 Projektplanung

### 3.1 Projektphasen

## 4 Praktische Realisierung

### 4.1 Theoretische Grundlagen CMUSphinx

### 4.2 Theoretische Grundlagen JavaFX

### 4.3 Einrichtung und Vorbereitung der Umgebung

### 4.4 Ausführung

#### 4.4.1 Modul 1 (Integration von CMUSphinx)

#### 4.4.2 Modul 2 (Erstellung der grafischen Oberfläche mit JavaFX)

## 5 Junit and Jenkins CI AutoTesting

## 6 Ergebnis

### 6.1 Fazit

## 7 Literaturverzeichnis, Anhang

## 1.1 Unternehmensprofil Auftraggeber

Die aicas GmbH ist ein 2001 gegründetes Technologieunternehmen, das sich der Entwicklung umfassender Softwarelösungen für verschiedene Bedürfnisse und Anforderungen widmet.

Es befindet sich derzeit im technologischen Herzen des Karlsruher Technologie-Park und setzt sich seit seiner Gründung für die Integration von Fachleuten aus verschiedenen Teilen der Welt ein, was einen multikulturellen und exotischen Reichtum in dem dort erlebten Arbeitsumfeld bietet.

Von Anfang an widmete sich aicas RealTime in der Softwareentwicklung und integrierte wesentliche Verbesserungen für verschiedene Anforderungen der anspruchsvollsten Kunden. Aufgrund der Effizienz und Qualität der Ergebnisbereitstellung stellte dies jedoch neue Herausforderungen und damit auch neue Kunden und Kunden potenzielle Bereiche der professionellen Erkundung:

- Landwirtschaft
- Avionik und Luft- und Raumfahrt
- Medizinische Industrie
- Industrielle Fertigung
- Elektronik
- Militär
- Automobil
- lot

Unabhängig davon, ob Sie ein Netzwerk industrieller IoT-Sensoren oder eine Flotte intelligenter Fahrzeuge aufbauen, bieten aicas-Softwarelösungen Sicherheit, Modularität und Portabilität für die anspruchsvollsten Anforderungen.

## 1.2 Mitwirkende Abteilungen

*Customer Delivery ist die Abteilung, die für die Lieferung der fertigen Version einer Entwicklungslösung an den Kunden zuständig ist. Es gibt drei weitere Teams, die die Hardware und Software integrieren. Sowohl das Verhalten als auch die Funktionalität der Software wird überwacht, bevor die endgültige Lieferung erfolgt, und funktionale Prototypen werden gestartet, um den Umfang und die Ergebnisse zu messen. Auf diese Weise wird das Endprodukt verbessert und Fehler sowie Inkompatibilitäts- oder Funktionsprobleme werden minimiert.*

Das Projekt wird in der Abteilung "Customer Delivery" durchgeführt. Die Abteilung besteht aus drei Teams. Dieses Projekt ist im "Peripherals Team" angesiedelt, in dem die Integration zwischen Embedded Hardware und Software stattfindet und deren Funktionen erweitert und getestet werden.

Im Rahmen des Projektes wird weiterhin Continuous Integration Software (Jenkins) und Hardware verwendet, welche vom "Platforms Team", ebenfalls Teil der "Customer Delivery", bereitgestellt wird.

Sowohl die Software als auch die Hardware für die Continuous-Integration. (Jenkins) werden weiterhin als Teil des Projekts verwendet, da der CI Prozess für die Erstellung von Abnahmetests relevant sind, die parallel im "Testing-Team" entwickelt wurden. Somit können die mögliche besser identifizieren Fehler in verschiedenen Build-Versionen der Software, und es wird auch erwartet, dass die Tests als integrale Phase der Projektarbeit entwickelt werden.

Durch die Integration folgend dem generellen Entwicklungsprozess wird die vollständige Integration des Projektergebnats in zukünftige Produkte ermöglicht. Es soll als Referenz für neue technologische Implementierungen bei aicas dienen und somit auch die Bedürfnisse potenzieller Kunden befriedigen.

## 2.1 Analyse der Ausgangssituation (Umfang)

Dieses Projekt besteht aus der Erstellung und Integration einer grafischen Oberfläche für 4 Demos, die mit der CMUSphinx-Bibliothek ausgeliefert werden. Diese wurden ausgewählt, um einen grundlegenden Sprachassistenten zu bilden, der die 4 Demos in einem einzigen Programm mit einer einheitlichen Oberfläche zusammenfasst.

Java wurde als Programmiersprache gewählt, da sie sehr gut portierbar ist und aicas seine Lösungen auf Java-Basis auf vielen verschiedenen Betriebssystemen anbietet. Weiterhin verfügt Java über einen robusten Code und seine grundlegende Funktionalität ermöglicht die Verwendung von einer großen Auswahl an Open-Source-Bibliotheken.

Für den Aufbau der grafischen Oberfläche wird JavaFX verwendet.

Der am häufigsten vorkommende Anwendungsfall wäre auf die IoT- und Automobilindustrie ausgerichtet. Mit der Spracherkennung können nicht nur die Lichter in einem Büro ein- oder ausgeschaltet werden, sondern es können auch grundlegende Funktionen oder Aufgaben eingerichtet werden, die das Fahren von Fahrzeugen über Sprachbefehle erleichtern. Durch Drücken einer einzigen Taste auf dem Touchscreen des Fahrzeugs kann der Benutzer oder Fahrer vorprogrammierte Sprachfunktionen verwenden, um den Wetterstatus und die aktuelle Uhrzeit zu ermitteln oder einfach zusätzliche Fahraufgaben festzulegen. Beispielsweise Einschalten des Radios, Wechseln der Sender, einschalten der Heizung oder Klimaanlage, Bewegen der Rückspiegel oder Aktivieren der Scheibenwischer, wenn es regnet.

Das Ziel des Projekts ist es, dem Benutzer die Möglichkeit zu geben, per Sprachbefehl zu kommunizieren und dem Programm eine möglichst natürliche Reaktion zu ermöglichen. Hierzu können die Parameter für die Audioein- und ausgabe sowie Grammatik- und Phonetikdateien eingestellt werden.

## 2.2 Lastenheft

Der folgende Auszug aus den Spezifikationen definiert die Anforderungen, die die zu entwickelnde Anwendung erfüllen muss. Die Anwendung berücksichtigt die Beteiligung der vertretenen Akteure im das 4.4.1 Modul 1 (Integration von CMUSphinx) und das 4.4.2 Modul 2 (Sammeln der grafischen Beschränkungen mit JavaFX) dargestellt durch ihr jeweiliges User-Uml-Diagramm.

### Anforderungen

Die folgenden Anforderungen werden in die grafische Lösung gestellt:

- Als Entwickler muss ich in der Lage sein, neue lokale phonetische und sprachliche Ressourcen zu konfigurieren und zu erstellen, da das Porogramm sie für die neue Schnittstelle und Ausführung von Demos benötigt.
- Als Entwickler muss ich in der Lage sein, den Inhalt der Dateien zu sehen und zu bearbeiten, die als lokale Ressourcen verwendet werden. Dazu wird eine Online-Software verwendet, um diese Elemente zu erstellen (Sphinx Knowledge Base Tool).
- Als Entwickler muss ich in der Lage sein, die jeweiligen Klassen innerhalb jeder Klasse aufzurufen, da eine Konfigurationsänderung so schnell wie möglich und jederzeit implementiert werden muss.
- Als Entwickler muss ich in der Lage sein, lokale Dateien oder Ressourcen zu löschen, da Beispieldateien oder erstellte Dateien gelöscht werden können.
- Als Entwickler muss ich optional zwei Klassen aufrufen können, eine Klasse für die Echtzeit-Audioerkennung oder eine Klasse für die lokale Audioerkennung über WAV-Audiodateien.
- Als Entwickler muss ich in der Lage sein, eine Hauptklasse zu erstellen, um alle den Demos entsprechenden Klassen aufzurufen und Ergebnisse daraus zu erhalten.
- Als Entwickler muss ich in der Lage sein, eine Hauptklasse zu erstellen, die die Application-Klasse erweitert, um alle von JavaFX bereitgestellten Optionen zu verwenden.
- Als Entwickler muss ich in der Lage sein, mehrere Elemente in einem Container zu erstellen und ihm Namen und Attribute zuzuweisen..
- Als Entwickler muss ich in der Lage sein, mit einem eventHandler den Durchgang von Objekten durch die verschiedenen Szenen zu steuern.
- Als Entwickler muss ich das Programm zum Zeitpunkt des Drückens von Tasten oder des Anzeigens von Fenstern und deren jeweiligen Inhalten problemlos zum Laufen bringen
- Als Entwickler möchte ich automatisierte Tests mit GitLab und Jenkins, damit die Tests zuverlässig und regelmäßig ausgeführt werden.
- Als Entwickler möchte ich in der Lage sein, die Kapselungsoptionen des Programms zu implementieren und so eine ausführbare Version zu erstellen. JAR-Executable
- Als Mitarbeiter möchte ich sowohl die Funktionsweise des Programms als auch das Verhalten der Funktionen in verschiedenen Systemen verstehen können, um festzustellen, ob es Hindernisse oder Schwierigkeiten gibt oder nicht.

## 3.1 Projektphasen

### 1 - Vorbereitung (3 Stunden)

- Ermittlung von und Beschreibung der Use-Cases (1 Stunde)
- Setup des Entwicklungsprojektes, Einbindung von CMUSphinx, JavaFX und JUnit (2 Stunden)

### 2 - Entwurfsphase (10 Stunden)

- User UML Diagramm (1 Stunde)
- Java UML Diagramm (1 Stunde)
- Flow Diagramm (2 Stunden)
- Struktogramm (2 Stunden)
- Entwurf der grafischen Oberfläche mit SceneBuilder und FXML (4 Stunden)

### 3 - Implementierung der Java-basierten Audioanbindung (18 Stunden)

- CMU-Sphinx-Parameterkonfiguration und Erstellung eines *Wörterbuchs*, *Vokabeln* und eines spezifischen *Sprachmodells* als notwendige Ressourcen für die Demos. (2 Stunden)
- Entwicklung der Mikrofonaufnahmefunktion mittels `javax.sound` und Entwicklung des Ladens von Audiodateien aus dem Dateisystem (`java.io/java.nio`) zur Ansteuerung der TranscriberDemo. (8 Stunden)
- Integration der Spracheingabe (Audiodatei, Mikrofon) in die 3 Varianten der DialogDemo: *Mathematische Operationen*, *Chatbot Dialog* und *Zurück zum Hauptmenü*. Die eigentliche Spracherkennung erfolgt über CMUSphinx, die Integration sowie die Erfassung und Überprüfung der Eingabeparameter sind Eigenentwicklungen. (2 Stunden)
- Integration der Sprecher-Erkennung einer bestimmten Spracheingabe (Audiodatei, Mikrofon) für die SpeakerIDDemo. Die eigentliche Spracherkennung erfolgt über CMUSphinx, die Parametrisierung über die Audiodaten ist eine Eigenentwicklung. (2 Stunden)
- Implementierung des Abgleichs zwischen einem Transkriptionsresultat und einer Audiodatei zur Kalibrierung der AlignerDemo. Die eigentliche Spracherkennung erfolgt über CMUSphinx, die Parametrisierung über die Audiodaten ist eine Eigenentwicklung. (2 Stunden)

Setup der Test-Automatisierung:

- Setup der Test-Automatisierung über die JUnit-Funktionalität von Jenkins. (2 Stunden)

### 4 - Implementierung der JavaFX Grafikschnittstelle (24 Stunden)

1. UI-Design ohne funktionale Implementierung (Eigenentwicklung):
  - Neuentwicklung eines Gerüsts (in JavaFX "Stage" genannt) für eine JavaFX Anwendung. (2 Stunden)
  - Hinzufügen aller GUI-Elemente durch JavaFX Code. Erstellung der benötigten Szenen und eines Hauptmenüs mit den benötigten Schaltflächen für die jeweiligen Demos. (5 Stunden)
2. Anwendungslogik der UI (Eigenentwicklung):
  - Entwicklung der Präsentation aller grafischen Resultate für jede Demo. (5 Stunden)
  - Entwicklung der Entgegennahme von Parametern inkl. Fehlerbehandlung. (4 Stunden)
  - Integration der Audiofunktionalität aus Arbeitspaket 2 in die UI-Schnittstelle. (8 Stunden)

## 5- Dokumentation der Projektarbeit und Erstellung der Präsentation (10 Stunden)

In diesem letzten Arbeitspaket werden alle Programmier- und Entwurfsarbeiten dieser Softwarelösung unter Berücksichtigung entstandener Probleme dokumentiert:

- Erstellung der Nutzerdokumentation (2 Stunden)
- Erstellung der Entwicklerdokumentation (2 Stunden)
- Erstellung der Projektdokumentation (6 Stunden)

## 4.1 Theoretische Grundlagen CMUSphinx

### Definition

*CMUSPhinix ist eine externe Spracherkennungsbibliothek, die vollständig von der CMU University in Java entwickelt wurde und ein Spracherkennungssystem bietet, das an verschiedene Anforderungen angepasst werden kann.*

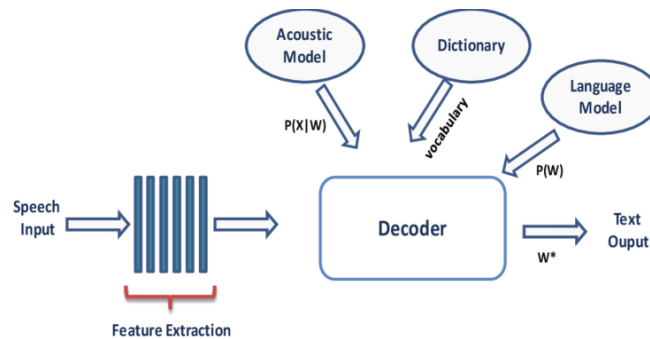
*Für dieses Projekt wird Sphinx Version 4 verwendet, eine vollständige, aktuelle und aktualisierte Version des Sphinx-Tools. Diese Version wird verwendet, weil sie ein flexibleres Framework zur Untersuchung der Spracherkennung bietet, da sie vollständig in der Programmiersprache Java geschrieben ist und Elemente enthält, die die Integration dieser Bibliothek erleichtern.*

### Funktion

Die Funktionen dieses externen Spracherkennungssystems sind mit der Erfüllung verschiedener Anforderungen verbunden, die sich in den Funktionen seiner Demos oder Beispiele widerspiegeln.



## Infografik zum Betrieb der Spracherkennung und ihrer verschiedenen Module



Bildquelle: [https://www.researchgate.net/publication/323878670\\_Amazigh\\_Speech\\_Recognition\\_System\\_Based\\_on\\_CMUSphinx](https://www.researchgate.net/publication/323878670_Amazigh_Speech_Recognition_System_Based_on_CMUSphinx)

### Basic Usage

Um mit CMUSphinx experimentieren zu können, müssen drei grundlegende Anforderungen berücksichtigt werden, die für die Durchführung der Tests erforderlich sind. Diese sind:

- Ressourcen und Repository (<https://oss.sonatype.org/#nexus-search;quick~cmusphinx>)
- Java-Kenntnisse (Java 1.8)
- Verwenden eine IDE (Eclipse)

*Die Schritte zum Vorbereiten der Arbeitsumgebung werden später im Umgebungskonfigurationsindex erwähnt.*

Sobald diese Anforderung erfüllt ist, können wir mit der erste Demonstration fortfahren:

## Transcriber-Demo Klasse, in der von CMUSphinx vordefinierte Methoden und Klassen für die Spracherkennung einhalten werden

```
import edu.cmu.sphinx.api.Configuration;
import edu.cmu.sphinx.api.SpeechResult;
import edu.cmu.sphinx.api.StreamSpeechRecognizer;

public class TranscriberDemo {

    public static void main(String[] args) throws Exception {

        Configuration configuration = new Configuration();

        configuration.setAcousticModelPath("resource:/edu/cmu/sphinx/models/en-us/en-us");
        configuration.setDictionaryPath("resource:/edu/cmu/sphinx/models/en-us/cmudict-en-us.dict");
        configuration.setLanguageModelPath("resource:/edu/cmu/sphinx/models/en-us/en-us.lm.bin");

        StreamSpeechRecognizer recognizer = new StreamSpeechRecognizer(configuration);
        InputStream stream = new FileInputStream(new File("resources/transcriber/10001-90210-01803.wav"));

        recognizer.startRecognition(stream);
        SpeechResult result;
        while ((result = recognizer.getResult()) != null) {
            System.out.format("Hypothesis: %s\n", result.getHypothesis());
        }
        recognizer.stopRecognition();
    }
}
```

Quelle: <https://github.com/cmusphinx/sphinx4/blob/master/sphinx4-samples/src/main/java/edu/cmu/sphinx/demo/transcriber/TranscriberDemo.java>

In dieser Klasse können wir die Verwendung von Elementen der Bibliothek beobachten, auf die wir im Folgenden näher eingehen werden:

- Configuration: In dieser Klasse werden die notwendigen Elemente festgelegt, um dem Programm sprachliche und grammatikalische Ressourcen zur Verfügung zu stellen, die für die Spracherkennung wesentlich sind.

Der Konfigurationsprozess ist einfach, jedoch müssen diese 4 Parameters berücksichtigt werden:

- Akustisches Modell: acoustic\_file\_folder
- Wörterbuch: file.dict
- Grammatik / Sprachmodell: file.gramm (optional)
- Sprachquelle: file.lm
- Jeder Parameter verwendet lokal eine Datei eines bestimmten Formates.
- Die Datei kann dann von der Erkennungsklasse verwendet werden

Zwei spezifische Klassen werden verwendet, um die oben genannten Dateien zu laden und den Analyseprozess vor der Erkennung zu starten:

- LiveSpeechRecognizer: Die Elemente dieser Klasse geben dem Programm die Möglichkeit, die Spracheingabe über ein Mikrofon in Echtzeit zu hören und auf diese Weise über Sprachbefehle eine Rückmeldung zwischen dem Benutzer und dem Programm zu erstellen
- StreamSpeechRecognizer: Diese Klasse bietet die Möglichkeit, den spezifischen Inhalt einer Audiodatei lokal zu interpretieren, sodass das Programm die Parameter und die Konfiguration verwenden kann, die als Werkzeuge für die Transkription und das Lesen festgelegt wurden.

## Demos

- Transcriber: Zeigt an, wie eine Datei transkribiert wird
- Dialog: Zeigt, wie ein Dialog mit einem Benutzer geführt wird.
- Speaker-ID: Identifikation der Sprecheranzahl
- Aligner: Demonstration des Zeitstempels von Audio bis Transkription

*Die Standardfunktionen jeder Demo werden verwendet, um unser Programm zu verbessern und so die Spracherkennung einer von Grund auf neu erstellten Oberfläche zu beleben. Es müssen jedoch zusätzliche Funktionen erstellt und in vorhandene Demos integriert werden.*

## 4.2 Theoretische Grundlagen JavaFX

### Definition

JavaFX ist eine Technologie, die auf der Java-Plattform basiert, um Webanwendungen mit Desktop-Anwendungsfunktionen anzubieten. Diese auf Java basierende Technologie ermöglicht das Erstellen von Desktop-Anwendungen für Mobiltelefone, Web, TV und viele andere Plattformen, die JRE- oder Java-Kompatibilität haben.

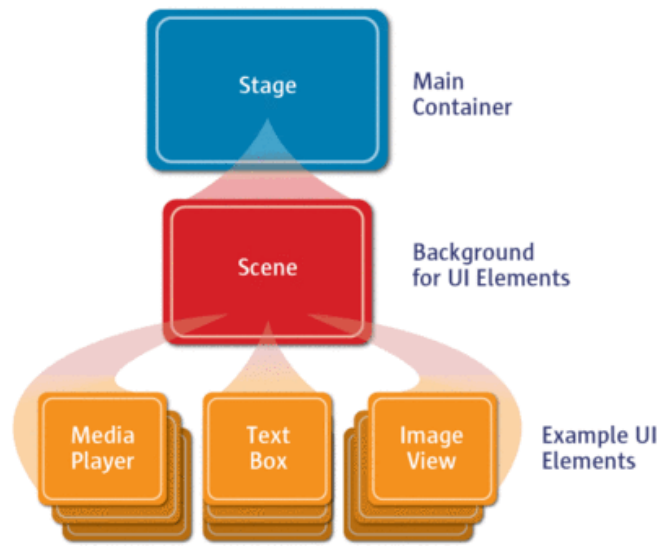
### Funktion

Das Erstellen eines JavaFX-Programms beginnt mit der *Application-Klasse*, von der alle JavaFX-Anwendungen ausgehen. Ihre Hauptklasse muss die Methode *launch()* aufrufen, die dann die Methode *init()* und dann die Methode *start()* aufruft, auf das Beenden der Anwendung wartet und dann die Methode *stop()* aufruft. Von diesen Methoden ist nur die *start()* Methode abstrakt und sollte überschrieben werden.

Die Stage-Klasse ist der JavaFX-Container der obersten Ebene. Wenn eine Anwendung gestartet wird, wird eine erste Phase erstellt und an die *Startmethode* der Anwendung übergeben. Die Stufen steuern die grundlegenden Eigenschaften des Fensters wie *Titel*, *Symbol*, *Sichtbarkeit*, *Größenänderung*, *Vollbildmodus* und *Dekorationen*.

Letzteres wird mit *Stage-Style* konfiguriert. Zusätzliche Stufen können nach Bedarf erstellt werden. Nachdem eine Stage konfiguriert und Inhalte hinzugefügt wurden, wird die *show ()* Methode aufgerufen.

## Offizielle Infografiken des Javafx und seiner Struktur



Bildquelle: <https://code.makery.ch/library/javafx-tutorial/>

## Basic Usage

Wenn wir das alles wissen, können wir ein minimales Beispiel schreiben, das ein Fenster in JavaFX startet:

### JavaFX-Beispiel – Container mit HelloWorld-Titel

```
import javafx.application.Application;
import javafx.stage.Stage;

public class Example1 extends Application
{
    public static void main(String[] args)
    {
        launch(args);
    }

    public void start(Stage theStage)
    {
        theStage.setTitle("Hello, World!");
        theStage.show();
    }
}
```

## Entwicklungsoptionen

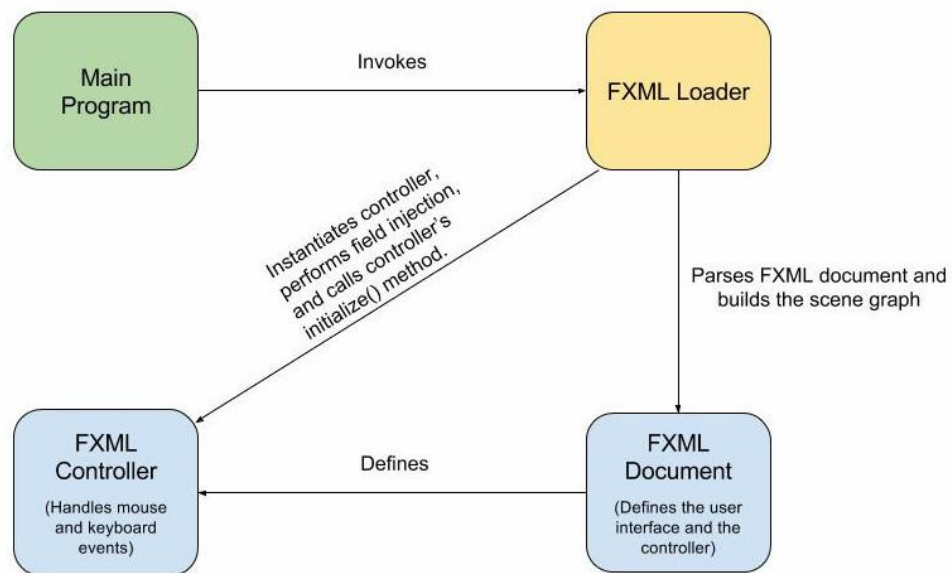
Abgesehen von dem umfangreichen Java-Code, der von der JavaFX-Bibliothek bereitgestellt wird, können wir die folgenden Optionen verwenden, um weiter mit dieser Technologie zu experimentieren und Demos

oder interessante Grafiklösungen zu entwickeln:

- **FXML:** JavaFX FXML ist ein XML-Format, mit dem wir JavaFX-GUIs ähnlich wie HTML-GUIs programmieren können. Daher können wir mit FXML unseren JavaFX-Designcode vom Rest des in Java entwickelten Anwendungscodes trennen. Auf diese Weise können wir sowohl den Entwurf des Codes als auch den Rest des Anwendungscodes bereinigen.

Andererseits kann FXML sowohl zum Erstellen des GUI-Entwurfs einer vollständigen Anwendung als auch nur eines Teils der GUI einer Anwendung verwendet werden, z. B. zum Entwerfen eines Teils eines Formulars, einer Registerkarte, eines Dialogfelds usw.

### Infografiken auf der Struktur und dem Betrieb von FXML



Bildquelle: <https://www.callicoder.com/javafx-fxml-form-gui-tutorial/>

FXML gibt uns eine schnelle und einfache Möglichkeit, UIS deklarativ zu definieren. Mit der SzeneBuilder-Tool-Unterstützung können Sie diese Art von Dateien auch automatisch generieren, was gut genug ist, um mit Leichtigkeit FXML zu schreiben.

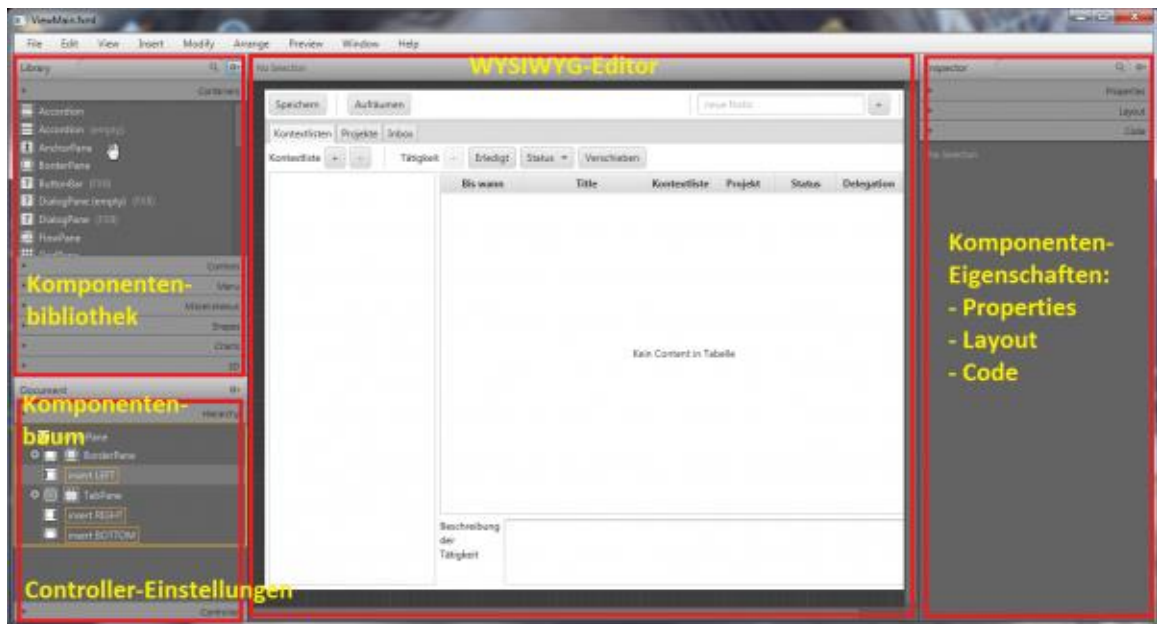
## FXML-Format – Demo, Container mit Button

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>

<VBox>
  <Label text="Hello world!"/>
  <Label text="This is a simple demo application."/>
  <Button text="Click me!"/>
</VBox>
```

- **Simulation in SceneBuilder:** Das Schreiben unserer GUI-Struktur in XML kann natürlicher sein als Java (insbesondere wenn wir mit HTML vertraut sind). Es ist jedoch immer noch nicht sehr praktisch für diejenigen, die mit dieser Sprache nicht vertraut sind. Die gute Nachricht ist, dass es ein offizielles Tool namens Scene Builder gibt, mit dem wir eine Benutzeroberfläche erstellen können. Einfach ausgedrückt ist es ein grafischer Editor für unsere GUI.



Bildquelle: <https://sopra.cs.tu-dortmund.de/wiki/infos/tutorials/javafx/scenebuilder>

Wie im Bild zu sehen ist, gibt es im Editor standardmäßig drei Hauptabschnitte:

- **Linker Abschnitt:** Hier werden die verfügbaren Komponenten angezeigt, die im mittleren Teil gezogen und abgelegt werden können. Dieser Abschnitt enthält auch die Hierarchie aller

Komponenten in der Benutzeroberfläche, sodass Navigation und Interaktion erleichtert werden können.

- **Zentraler Abschnitt:** In diesem Abschnitt werden die basierend auf der FXML-Datei vorgenommenen Änderungen gerendert
- **Rechter Abschnitt:** Hier befindet der aktuelle Komponenteninspektor. In diesem Abschnitt könnte man verschiedene Eigenschaften der aktuell ausgewählten Komponente bearbeiten. *Jede Komponente, die aus der Mitte der Hierarchie ausgewählt wird, wird im Inspektor angezeigt.*

## 4.3 Einrichtung und Vorbereitung der Umgebung

### Verwendete Ressourcen

#### Hardware:

- Büroarbeitsplatz
- Laptop
- Tastatur
- Maus
- Zweiter Monitor
- Telekommunikationsanlage (Wifi- Ethernet)
- Headphones

#### Software:

- Linux Centos 7 - Betriebssystem
- Mozilla Firefox - Web Browser für Untersuchung und Ausführung der Assistant-Demo (Öffnen einen bestimmten Browser mit Sprachbefehlen)
- JDK 8 - Neueste Version von Java
- EclipseIDE 12-2020 - Entwicklungsumgebung Java und JavaFX
- Maven 4.0.0 - Tool zur Verwaltung und Erstellung von Java-Projekten
- Jenkins 2.2.7 - Open Source-Automatisierungsserver
- JavaFX 11.0.2 - Framework zur Erstellung plattformübergreifender Java-Applikationen
- CMUSphinx 4 - Spracherkennungssysteme

Als nächstes werde ich Einzelheiten zu allen Schritten angeben, die zum Konfigurieren einer Umgebung verwendet werden, die der Implementierung und Entwicklung der jeweiligen grafischen Oberfläche für die Ausführung der Demos förderlich ist:

1. **Die externe Ressourcen wurde heruntergeladen**
  - CMUSphinx- resources (APIs)
  - JavaFx-resources (APIs)
2. **Die spezifische Eclipse IDE wurde konfiguriert**

- Maven-Projekterstellung
  - Integrieren der heruntergeladenen APIs als externe Bibliotheken in die Build-Path-Option des Projekts
3. **Die spezifische Gitlab repository wurde konfiguriert**
    - Synchronisieren des vorhandenen Master-Repository mit dem Ordner in dem sich unser Projekt befindet
    - Erstellen des ersten Commits und hochladen der Änderungen
  4. **Die spezifische Jenkins Server wurde konfiguriert**
    - Schaffung eines neuen Jobs als Freestyle-Projekt
    - Synchronisieren unseres Repository mit dem neu erstellten Job
    - Einfügen eines Build-Schrittes
    - Testen mit Hilfe von Maven durch ausführen der entsprechenden Befehle
  5. **Entwicklung**

In dieser letzten Phase werden die ersten Verhaltenstests des Programms unter Verwendung der Bibliotheken und vorgegebenen Funktionen in den Demos entwickelt.

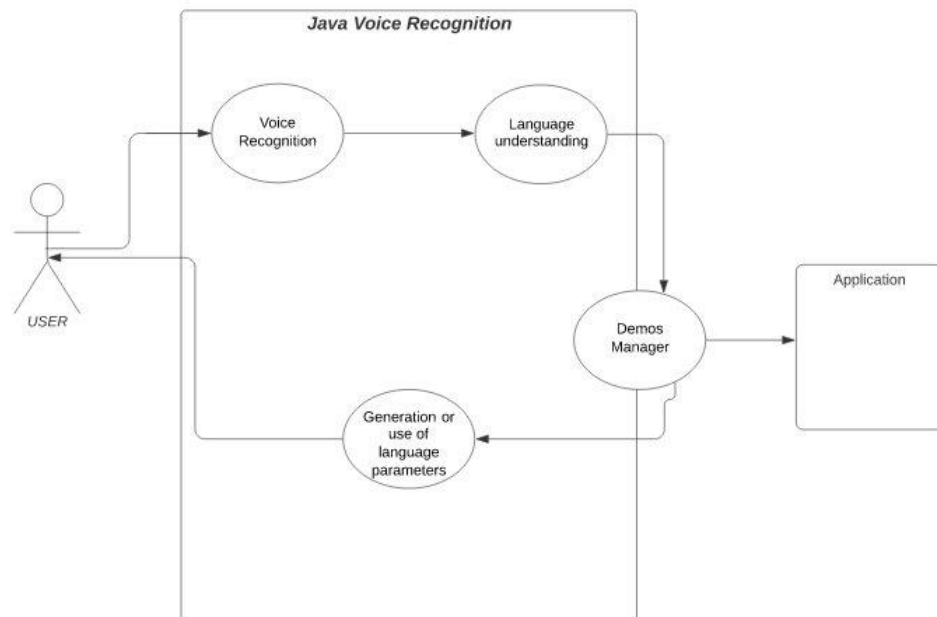
#### 4.4.1 Modul 1 (Integration von CMUSphinx)

Das Folgende ist eine Java-basierte Softwarelösung, die die externe CMUSphinx-Bibliothek verwendet. Dieses Programm ist vollständig synchronisiert, um mithilfe spezifischer Sprachbefehle, die für diese Demonstration konfiguriert wurden, mit den verschiedenen Elementen zu interagieren.

Die Operation basiert ohne weiteres auf der Integration von Java-Klassen, die den 4 von der Bibliothek bereitgestellten Demos entsprechen.



## Benutzer-UML-Spracherkennungsdiagramm mit CMUSphinx



Sowohl die Grundstruktur als auch die Funktionen dieses Programms, die von der CMUSphinx-Bibliothek bereitgestellt werden, bestehen aus den folgenden Modulen:

- **Spracherkennung:** Hier werden die Standardmethoden von CMUSphinx verwendet, um den Erkennungsprozess zu starten oder zu stoppen.

### Java-Klasse - LiveSpeechRecognizer

```
// RealTime Speech Recognition with the LiveSpeechRecognizer Class
LiveSpeechRecognizer recognizer = new LiveSpeechRecognizer(configuration);
```

Quelle: <https://github.com/cmusphinx/sphinx4/blob/master/sphinx4-samples/src/main/java/edu/cmu/sphinx/demo/transcriber/TranscriberDemo.java>

Diese Klasse bietet die Möglichkeit, sich automatisch und in Echtzeit für Spracherkennungsmethoden zu entscheiden.

## Methoden der Klasse Java Livenspeechrecognizer - Start oder Stop

```
// Start recognition process pruning previously cached data.
recognizer.startRecognition(true);
SpeechResult result = recognizer.getResult();

// Pause recognition process. It can be resumed then with startRecognition(false).
recognizer.stopRecognition();

while ((result = recognizer.getResult()) != null) {
    System.out.format("Hypothesis: %s\n", result.getHypothesis());
}

// Pause recognition process. It can be resumed then with startRecognition(false).
recognizer.stopRecognition();
}
```

Quelle: <https://github.com/cmusphinx/sphinx4/blob/master/sphinx4-samples/src/main/java/edu/cmu/sphinx/demo/transcriber/TranscriberDemo.java>

Durch Aufrufen dieser Methode können wir den Spracherkennungsprozess entweder starten oder stoppen

- **Sprachgenerierungs- und Synthesemodul:** Es werden bestimmte Parameter verwendet, die von der Konfigurationsklasse bereitgestellt werden.

Jeder Parameter verfügt über eigene Dateien, die zum Starten des Sprachanalyseprozesses erforderlich sind.

## Java-Konfigurationsklasse - Parameter einrichten.

```
Configuration configuration = new Configuration();

configuration.setAcousticModelPath("resource:/edu/cmu/sphinx/models/en-us/en-us");
configuration.setDictionaryPath("resource:/edu/cmu/sphinx/models/en-us/cmudict-en-us.dict");
configuration.setLanguageModelPath("resource:/edu/cmu/sphinx/models/en-us/en-us.lm.bin");
```

Quelle: <https://github.com/cmusphinx/sphinx4/blob/master/sphinx4-samples/src/main/java/edu/cmu/sphinx/demo/transcriber/TranscriberDemo.java>

Indem diese Sprachbefehle in der vorherigen Methode als verwendbare Parameter festlegen, können wir mit dem Programm und seinen verschiedenen Optionen interagieren.

Bedingung für die Ausführung von Sprachbefehlen exit und back - while, if und break

```
recognizer.startRecognition(true);
while (true) {
    String voiceCommand = recognizer.getResult().getHypothesis();
    if (voiceCommand.equals("exit")
        || voiceCommand.equals("back"))
        break;
    else
        System.out.println(voiceCommand);
}
recognizer.stopRecognition();
}
```

Wenn der Sprachbefehl "beendet" oder "zurück" ist, wird das Programm beendet, andernfalls wird die Erkennung fortgesetzt.

- **Semantische Analyse oder Sprachverständnis:** Für dieses Modul werden zwei Überprüfungsmethoden verwendet: *getResult()* und *getHypothesis()*. Das Ergebnis, das dem oben festgelegten Sprachbefehl entspricht, wird mit der Standardhypothesenmethode verglichen. Wenn sie übereinstimmen, wird die Standardaktion ausgeführt.

Lokale Variable VoiceCommand, die die Ergebnisse der Eingabe hält und sie mit den vorkonfigurierten Sprachbefehlen vergleicht.

```
String voiceCommand = recognizer.getResult().getHypothesis();
if (voiceCommand.equalsIgnoreCase("open browser")) {
    Runtime.getRuntime().exec("firefox www.google.com");
    System.out.println("The Browser is Opened..");
} else if (voiceCommand.equalsIgnoreCase("close browser")) {
    Runtime.getRuntime().exec("pkill firefox");
    System.out.println("The Browser is Closed");
}
break;
```

*In diesem Beispiel wird geprüft, ob das Wort des Sprachbefehls mit der Hypothese übereinstimmt, und je nach Ergebnis wird ein Browser geöffnet oder geschlossen.*

- **Demos-manager:** Für dieses Modul wurde eine Hauptklasse mit denselben Spracherkennungsfunktionen erstellt, die die Interaktion mithilfe von Sprachbefehlen mit den verschiedenen Demos ermöglicht.

#### Eclipse Ide Konsolenausgabe

```
Voice Command is WAKE UP
```

---

```
Voice Recognition Demos Menu
```

```
Option 1: Transcriber  
Option 2: Assistant  
Option 3: SpeakerID  
Option 4: Aligner  
Option 5: exit
```

---

```
Pls choose one of the options by voice:
```

---

*Jede der Hauptmenüoptionen kann durch Sprachbefehle gesteuert werden, da die Anfangsbedingung erfüllt ist und die `InitDemo()` Methode aufgerufen wird, in der alle Konfigurationsparameter sowie die Erkennungsklassen und methoden gekapselt sind. Die Ergebnisüberprüfung bleibt daher aktiv Dies ist zu jeder Zeit von Vorteil, wenn Multiprozesse verwaltet werden.*

#### Rec-Modus von TranscriberButton-Klasse:

Eine Funktion, die implementiert werden musste, entspricht der *RecDemo-Klasse* innerhalb der *Transcriber-Klasse*, mit der Sie die Aufzeichnung für einen bestimmten Zeitraum starten können. Die resultierende Audiodatei kann von der *StreamSpeechRecognizer-Klasse* verwendet werden, um Ressourcen lokal zu verwenden.

#### Zielpfad für aufgezeichnete Dateien der Rec-Klasse

```
// path of the wav file  
File wavFile = new File("resources/transcriber/newFile.wav");
```

```
InputStream stream = new FileInputStream(new File("resources/transcriber/newFile.wav"));
```

*Pfad, der von der Transkribentenklasse verwendet wird*

#### Dialog-Modus von AssistantButton-Klasse:

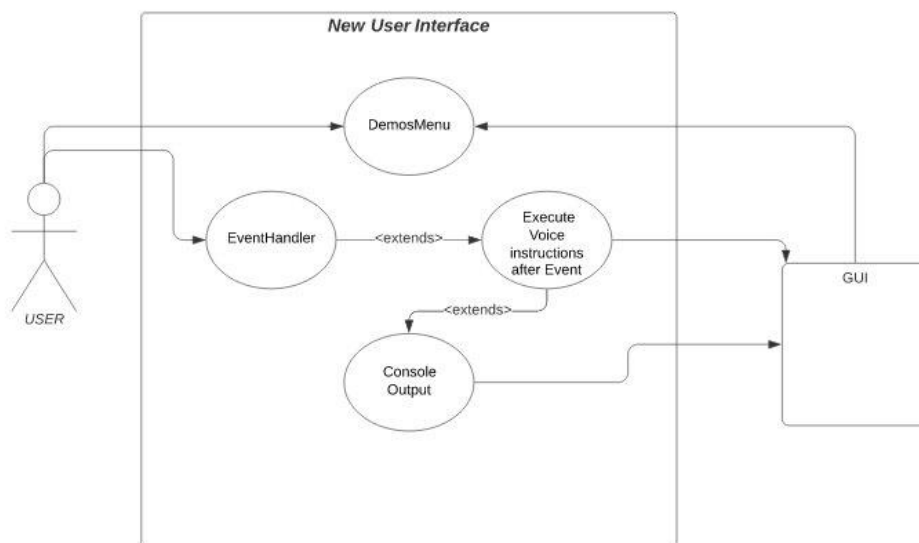
Assistant-Button ist eine CMUSPhinxDemo-Klasse, die 4 Anwendungsfälle liefert. Eine davon entspricht der neuen Funktion, die für diese Demo erstellt wurde, dem Dialogmodus, der in die neue Dialog-Demo-Klasse geladen wurde.

### 4.4.2 Modul 2 (Erstellung der grafischen Oberfläche mit JavaFX)

Da wir unser Programm bereits in Java entwickelt haben, müssen wir nur eine einfache grafische Oberfläche erstellen, um den Inhalt zu ordnen und die Verwendung der verschiedenen Abschnitte des Programms zu vereinfachen.

*Die Entwicklung dieser Lösung bietet dem Benutzer nur die Möglichkeit, bis zu einem gewissen Grad mit der grafischen Oberfläche zu interagieren und dann mithilfe der vorgegebenen Sprachbefehle das Interaktionssystem mit den anderen Elementen des Menüs weiter voranzutreiben.*

#### Benutzer-UML-Spracherkennungsdiagramm mit JavaFX



In diesem Diagramm sehen wir, dass es 2 Ausführungsmodule gibt, die mit der grafischen Oberfläche interagieren, auf die wir im Folgenden näher eingehen werden:

- **Demosmenü:** Für dieses Modul wurde eine einfache, aber effektive grafische Oberfläche erstellt, mit der der Benutzer die Werkzeuge zum Navigieren zwischen den einzelnen Menüelementen (Demos) erhalten kann.
- **Ereignishandler:** In diesem Modul werden die Aktionen und Ereignisse programmiert, die sich auf die Struktur und das Diagramm beziehen, die zuvor in der DemosMenu-Klasse beobachtet wurden. Zu diesem Zweck haben wir jedem Element auf sehr geordnete Weise eine Hierarchie zugewiesen, die auf *Stage*, *Scene*, *Layout*, *Label* und *Button* basiert. Um den Schaltflächen ein Ereignis zuzuweisen, wird die `setOnAction`-Methode verwendet, mit der dem Prozess `EventHandler`-Funktionen zugewiesen werden können.

Dies ist der erste Schritt unserer grundlegenden hierarchischen Struktur. Wenn wir jedoch die Aktion beim Drücken der Taste `btnTranscriber` analysieren, gelangen wir zu einer anderen Szene, die dem Untermenü `TranscriberButton` entspricht.

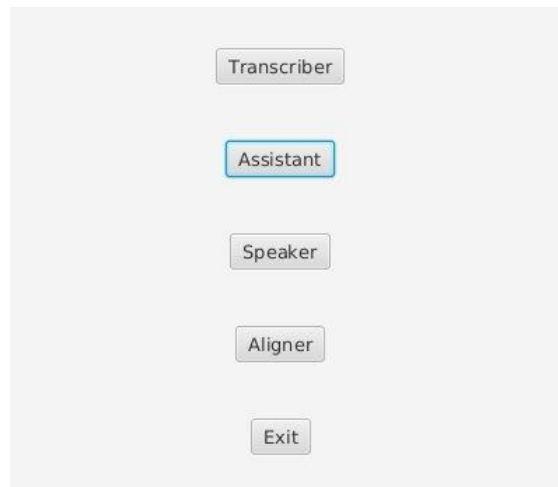
- Lambdas werden verwendet, *um Aktionen einfacher und effektiver zu verketteten, und jedem Untermenü wird eine eigene Szene zuzuweisen.*
  - in jedem Szenenterm die Mess- und Positionierungsparameter durch das entsprechende Layout definiert werden.
  - Als Option, die in die Szene integriert ist, können wir mit `btnBack` zum Hauptmenü zurückkehren oder das Programm mit `btnExit` beenden
- **Sprachbefehle und Ausgabekonsole:** Wenn eine Schaltfläche gedrückt wird, wird direkt in der Aktion ein indiziertes Ereignis generiert, mit dem eine bestimmte Klasse oder Methode aufgerufen werden kann. In diesem Fall wird die `AssistantDemo` Klasse aufgerufen.

#### ActionHandler - Aktion der Assistententaste zum Starten der spezifischen Szene

```
//Assistant Button action handler
btnAssistant.setOnAction(e -> {
    if (! Window.isShowing()) {
        btnAssistant.setOnAction(evn -> Window.setScene(myScene));
        try {
            AssistantButton.main(null);
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
});
```

Dies ist das Beispiel des `AssistantButton`-Menüs, das von einer Aktion verwaltet wird. Bei der Ausführung wird das Konsolenmenü angezeigt und die Spracherkennung gestartet.

Einfache grafische Benutzeroberfläche mit Schaltflächen, die den Demos entsprechen



VoiceMenu ohne GUI - Ausgabe über die Eclipse-Konsole

---

AssistantButton Menu

---

Option 1: Digit  
Option 2: Dialog  
Option 3: Bank  
Option 4: Back

---

Pls choose one of the options by voice:

---

## 5 Junit and Jenkins CI AutoTesting

In diesem Abschnitt wurde ein dedizierter Server für automatische Tests mit einer funktionierenden Jenkins-Installation konfiguriert. Zunächst wurden Unit-Test-Tests von Eclipse IDE bezüglich der verschiedenen Methoden und Klassen durchgeführt, deren Funktionalität getestet werden musste.

Als nächstes wurde das Projekt in ein dediziertes Gitlab-Repository hochgeladen. Schließlich wurde aus Jenkins ein Job im Pipeline-Stil erstellt, und mithilfe einer vorherigen Konfiguration der Anmeldeinformationen ein geplanter automatischer Test festgelegt werden.

## Automatische Unit-Test-Ausführung von Jenkins Project mit Maven und CI-Synchronisation mit Gitlab – Jenkins console output

- Für diesen Prozess wurde in Eclipse IDE ein Maven-Projekt erstellt, in dem die Klassen und die zur Ausführung des Tests erforderlichen Ressourcen gruppiert wurden, z. B. die Audiodateien sowie die für die Konfiguration erforderlichen Grammatik- und Sprachdateien
- Im Testabschnitt wurde eine grundlegende Testklasse erstellt, die die Hauptklasse TranscriberDemo verwendet und ihre Hauptmethode ausführt
- Bei der Ausführung des Tests wird die Klasse ausgeführt und am Ende ihrer Ausführung wird grün zurückgegeben. Dies bedeutet, dass der Test korrekt und fehlerfrei ausgeführt wurde
- Auf der anderen Seite erstellen wir einen Ordner, in dem wir unser Projekt speichern, und führen dann die Git-Befehle aus, die zum Synchronisieren und Übertragen der Dateien und Ordner des Projekts über die Befehlskonsole in das offizielle Repository erforderlich sind.
- Auf der Seite von Jenkins wird ein neuer Pipeline-Typ Jobs erstellt, in dem die Synchronisation mit dem spezifischen Repository über die von Jenkins angegebenen Optionen hergestellt wird. (Benutzer-, Kennwort- und Repository-Anmeldeinformationen sind erforderlich).
- Abschließend erstellen wir über Pipeline-Code die neuesten lokalen JDKPath- und Javapath-Variablen, damit das Programm bei der Übermittlung des Ergebnisses keine Ausführungs- oder Kompatibilitätsfehler auslöst. Außerdem legen wir die Parameter fest, die für die Syntax erforderlich sind, um eine Reihe von Schritten auszuführen, die es uns ermöglichen, unsere erste erfolgreiche Ausgabe zu erhalten.

*Durch die Optionen für die periodische Erstellung können wir von Zeit zu Zeit eine automatische Ausführung des Tests einrichten. In diesem Fall verwenden wir die Funktion der Ausführung alle 30 Minuten, um dessen Funktion zu überprüfen.*

*Der JUnit-Test wurde für die Klassen und Methoden durchgeführt, die lokal mit Audiodateien verwendet werden, z. B.: Transcriber, Aligner und SpeakerID. Die Tests sind nicht kompatibel mit der LiveSpeechRecognizer Klasse und ihren Methode.*



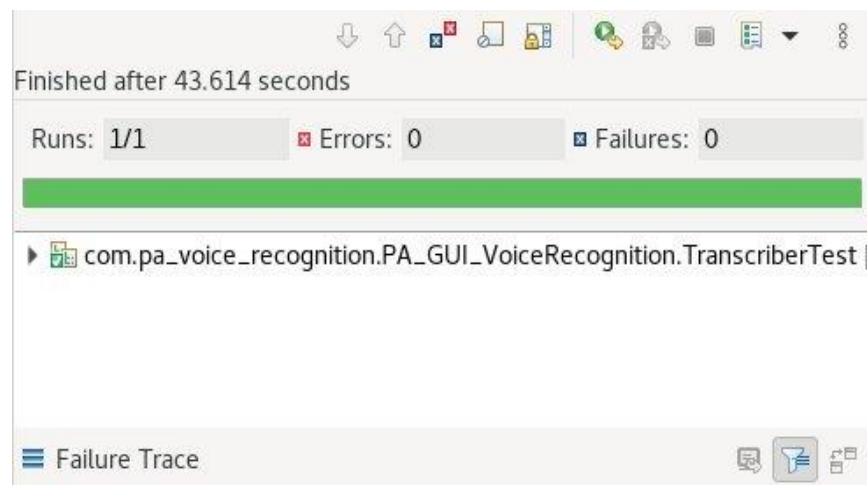
### Junit Simple-Test von Transcriber Demo

```
2
3+ import static org.junit.Assert.assertTrue;
7
8- /**
9  * Unit test for Transcriber Demo.
10 */
11 public class TranscriberTest
12 {
13-     /**
14      * Rigorous Test :-)
15      * @throws Exception
16      */
17
18-     @Test
19     public void TranscriberDemo() throws Exception
20     {
21
22         TranscriberDemo td = new TranscriberDemo();
23         td.main(null);
24
25     }
26
27 }
```

Die Transcriber-Demo wurde als Beispiel verwendet, um die Komponententests zu entwickeln, deren erforderliche Funktionalität die Spracherkennung über die *StreamSpeechRecognizer*-Klasse ist, die aufgerufen wird, um lokale Audiodateien im WAV-Format zu verwenden.

Durch Aufrufen der Klassen-Transcriber-Demo werden die Klasse und ihre Dateiverwendungsfunktionen lokal ausgeführt, was zu einer positiven Ausführung führt. Durch diese Ausführung wird die Funktionsweise der verschiedenen Klassen überprüft, wenn Ressourcen lokal verwendet werden.

### Ausgabe der Junit-Testausführung von Eclipse



### Ausgabe der Junit-Testausführung vom Befehlsterminal

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 42.075 s
- in com.pa_voice_recognition.PA_GUI_VoiceRecognition.TranscriberTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 44.285 s
[INFO] Finished at: 2021-04-26T09:14:35+02:00
[INFO] -----
```

### Ausgabe der Junit-Testausführung von Jenkins UI - Konfigurations-panel

The screenshot shows the Jenkins UI. On the left, the 'Build History' panel lists three builds: #680 (Apr 26, 2021 2:33 PM), #679 (Apr 26, 2021 2:03 PM), and #678 (Apr 26, 2021 1:33 PM). On the right, the 'Stage View' panel displays 'Average stage times: 13s' and 'Average full run time: ~14s'. A button on the right says 'Checkout the Git Repository and run the internal Junit Tests'.

### Ausgabe der Junit-Testausführung vom Jenkins-output-Befehls-console

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 7.503 s - in
com.pa_voice_recognition.PA_GUI_VoiceRecognition.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.052 s
[INFO] Finished at: 2021-04-26T14:33:14+02:00
[INFO] -----
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

## 6.1 Fazit

Nach Recherchen und Experimenten mit dieser von Grund auf neu erstellten grafischen Oberfläche können wir den Schluss ziehen, dass die Integration von Sprachbefehlsfunktionen auf der Basis der CMUSphinx-Bibliothek nicht nur zusätzliche Interaktionsfunktionen bietet, sondern auch die Benutzerfreundlichkeit zwischen Benutzer und Programm erheblich verbessert.

Auf einer Seite gibt es verschiedene Verbesserungsoptionen, die wir festlegen können, wenn wir die Entwicklung einer grafischen Oberfläche sowie alle Entwurfsoptionen und visuellen Effekte, die die JavaFX-Bibliothek bietet, untersuchen und vertiefen.

Andererseits ist diese Schnittstelle einfach zu entwickeln und basiert auf einem primitiven Java-Code. Durch die Integration beider Bibliotheken wird jedoch ein zukunftsweisend Tool zur Arbeitsunterstützung erhalten, das wir auf jeden Arbeitsbereich oder Anwendungsfall anwenden können. In medizinischen oder Forschungsbereichen kann diese grafische Lösung Usability-Probleme für Menschen mit Behinderungen lösen und uns Multitasking-Funktionen bieten, die zur Verbesserung der Qualität unserer täglichen Arbeit erforderlich sind.

*Dies ist das grafische Ergebnis der Entwicklung der Lösung für dieses Projekt. Derzeit ist die Navigation nur mit den Schaltflächen in der ersten Szene der GUI möglich, da die Synchronisation mit dem Thread für jede durch die Schaltflächen festgelegte Aktion verwaltet werden muss, um die Navigation fortzusetzen. Ein Teil der ursprünglich im 2.2 Lastenheft-Index geplanten Projekterwartungen, die mit den Event-Handlern zu tun hatten, konnte jedoch nicht erfüllt werden, da dies eine parallele Ausführung beider externer Bibliotheken erforderte, sodass die Integration beider externer Bibliotheken nicht möglich war. Fehlermeldungen oder Fenster, die den Inhalt jeder Klasse anzeigen.*

Leider konnten aus zeitlichen Gründen keine weiteren Fortschritte bei der Implementierung des Threads erzielt werden, da dies auch nicht Teil des Projektziels war. Die Umsetzung von Thread würde dem Projekt jedoch einen positiven Impuls geben, weshalb es perfekt als Verbesserungsvorschlag für die Zukunft verwendet werden könnte.

In Bezug auf die Stundenverteilung wurde auch eine interessante Änderung festgelegt, da zunächst 65 Stunden Gesamtentwicklung festgelegt wurden, die der Realität gegenübergestellt wurden und der Entwicklung von Phasen wie Dokumentation, Implementierung von CMUSphinx und Verbesserungen für die Schnittstellenentwicklung 5 zusätzliche Stunden hinzufügten. Ausführlich erläutert in Abschnitt A.1 Detaillierte Zeitplanung entsprechend Anhang.

Der Einsatz dieser Technologie kann viele Bereiche abdecken, entweder als intelligenter Sprachassistent, als ausführbare Desktop-Anwendung oder als wissenschaftliche und technologische Lösung für Forschungsfälle im medizinischen Bereich. Die Verwendung der Kombination beider Technologien war sehr zufriedenstellend, was mich intensiv motiviert, beide integrierten Technologien weiterhin zu verwenden, um die Entwicklung neuer Verbesserungsmöglichkeiten zu vertiefen und so das Ergebnis anzupassen, um ein viel größeres Ziel zu erreichen. "Verbesserung der Lebensqualität der Menschen".

## 7 Literaturverzeichnis

- **Offizielle JavaFX-Website**

[\*https://openjfx.io/\*](https://openjfx.io/)

- **GUI-Programmierblog und Forum**

[\*https://code.makery.ch/\*](https://code.makery.ch/)

- **Diagramme**

[\*https://lucid.app\*](https://lucid.app)

- **SceneBuilder**

[\*https://gluonhq.com/products/scene-builder/\*](https://gluonhq.com/products/scene-builder/)

- **Offizielle CMUSphinx-Website**

[\*https://cmusphinx.github.io/\*](https://cmusphinx.github.io/)

- **Sphinx Knowledge Base Tool**

[\*http://www.speech.cs.cmu.edu/tools/lmtool-new.html\*](http://www.speech.cs.cmu.edu/tools/lmtool-new.html)

- **Jenkins - offizielle Dokumentation**

[\*https://www.jenkins.io/doc/tutorials/\*](https://www.jenkins.io/doc/tutorials/)

Anhang:

A.1 Detaillierte Zeitplanung

A.2 Entwicklerdokumentation

A.2.1 Entwicklerdokumentation - Fehlerdokumentation

A.3 UML-Klassen, Flow und Nassi-Diagramm

A.4 Benutzerdokumentation

## A.1 Detaillierte Zeitplanung

Das Projekt besteht aus zwei Kernkomponenten, der Audio-Anbindung und der grafischen Schnittstelle. Es wird in fünf Entwicklungsphasen unterteilt deren Entwicklungsdauer den maximalen Zeitrahmen von 70 Stunden nicht überschreitet.

<b>1 - Vorbereitung</b>	<b>3 H</b>
Ermittlung von und Beschreibung der Use-Cases	1 H
Setup des Entwicklungsprojektes, Einbindung von CMUSphinx, JavaFX und JUnit	2 H
<b>2 - Entwurfsphase</b>	<b>10 H - 2 H</b>
User UML Diagramm	1 H
Java UML Diagramm	1 H
Flow Diagramm	2 H
Struktogramm	2 H
Entwurf der grafischen Oberfläche mit SceneBuilder und FXML	4 H - 2 H
<b>3 - Implementierung der Java-basierten Audioanbindung</b>	<b>18 H + 3 H</b>
CMU-Sphinx-Parameterkonfiguration	2 H
Entwicklung der Mikrofonaufnahmefunktion mittels javax.sound	8 H + 2
Integration der Spracheingabe (Audiodatei, Mikrofon)	2 H
Integration der Sprecher-Erkennung einer bestimmten Spracheingabe (Audiodatei, Mikrofon)	2 H
Implementierung des Abgleichs zwischen einem Transkriptionsresultat und einer Audiodatei	2 H
Setup der Test-Automatisierung über die JUnit-Funktionalität von Jenkins	2 H + 1 H
<b>4 - Implementierung der JavaFX Grafikschnittstelle</b>	<b>24 H + 3 H</b>
UI-Design ohne funktionale Implementierung	2 H
Hinzufügen aller GUI-Elemente durch JavaFX Code	5 H
Anwendungslogik der UI	5 H + 1 H
Entwicklung der Entgegennahme von Parametern inkl. Fehlerbehandlung	4 H
Integration der Audiofunktionalität aus Arbeitspaket 2 in die UI-Schnittstelle	8 H + 2
<b>5 - Dokumentation der Projektarbeit und Erstellung der Präsentation</b>	<b>10 H + 1 H</b>
Erstellung der Nutzerdokumentation	2 H
Erstellung der Entwicklerdokumentation	2 H + 1 H
Erstellung der Projektdokumentation + Präsentation PowerPoint	5 H + 1 H
<b>Gesamt</b>	<b>70 H</b>

## A.2 Entwicklerdokumentation

Die Entwicklerdokumentation soll einige Erklärungen für andere Entwickler enthalten, die an diesem Projekt arbeiten oder arbeiten werden, um so die Implementierung und Ausführung des Programms und seiner Funktionen zu erleichtern. Die Entwicklerdokumentation wird als Anleitung und Screenshots des Quellcodes zusammen mit den Kommentaren der entsprechenden Funktionen angezeigt. Dieser Prozess wurde durchgeführt, nachdem die Implementierungsphase

mit CMUSphinx abgeschlossen und eine grafische Oberfläche mit JavaFX erstellt wurde, die dem Leser einen vollständigen Überblick über die allgemeine Entwicklungsphase gibt.

In den folgenden Schritten werden die Optionen beschrieben, die der Entwickler beim Zugriff auf das für diese Demonstration erstellte Programm hat:

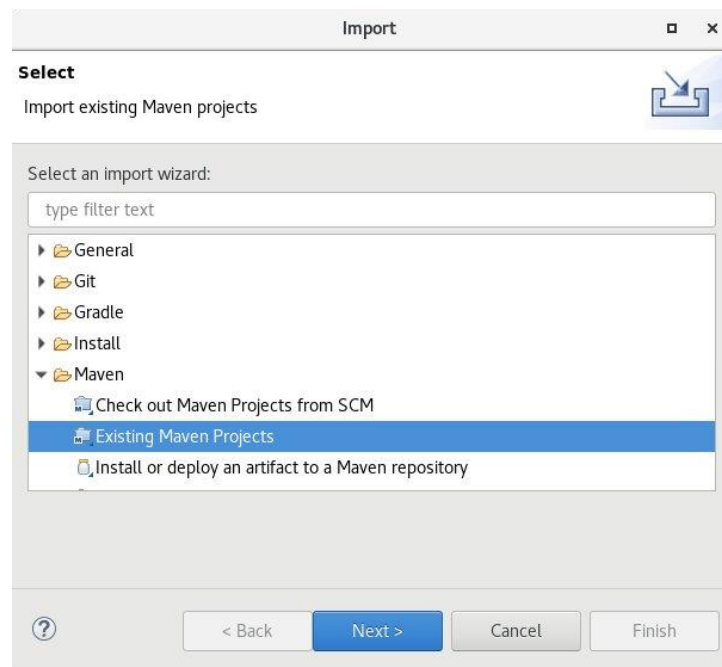
1. Erstellung einen Ordner, der das Projekt enthält

```
bash-4.2$ mkdir NewProject
```

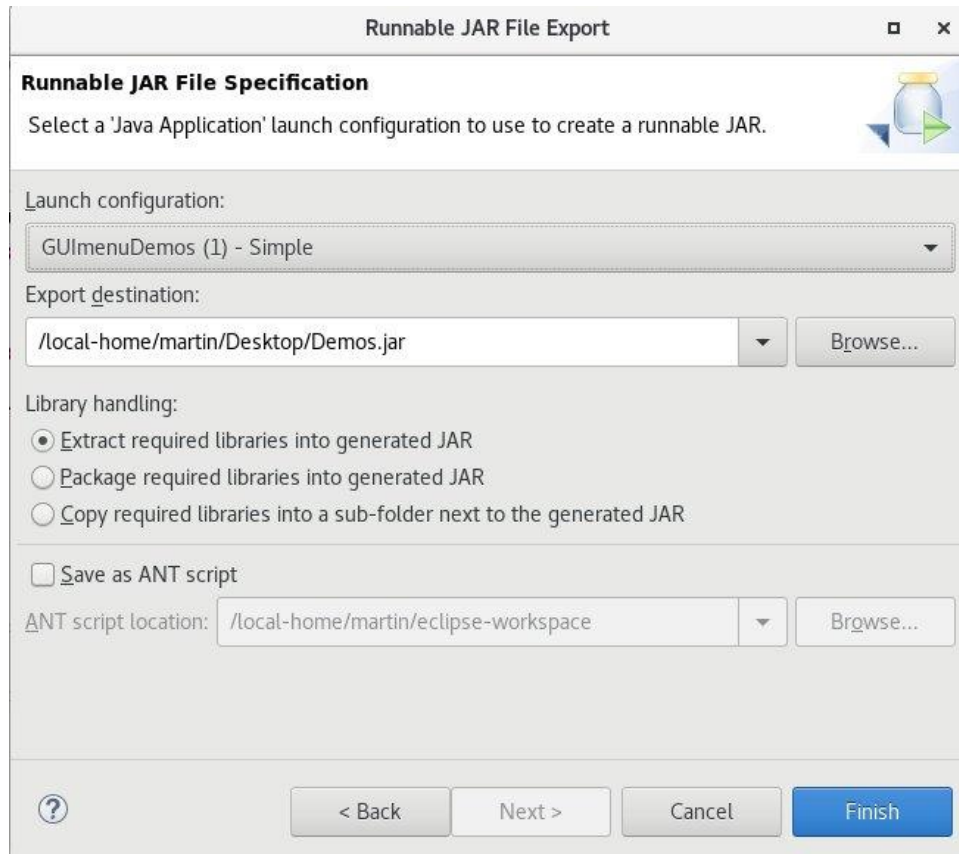
2. Klonen in den Stammordner erstellt das spezifische Repository

```
bash-4.2$ cd NewProject
bash-4.2$ git clone https://gitrepos.aicas.de/blauth/pa-voice.git
Cloning into 'pa-voice'...
Username for 'https://gitrepos.aicas.de': martin
Password for 'https://martin@gitrepos.aicas.de':
remote: Enumerating objects: 34, done.
remote: Counting objects: 100% (34/34), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 417 (delta 17), reused 0 (delta 0), pack-reused 383
Receiving objects: 100% (417/417), 73.13 MiB | 4.29 MiB/s, done.
Resolving deltas: 100% (111/111), done.
bash-4.2$ cd pa-voice
```

3. Durchsuchen den Inhalt über die Befehlskonsole oder importieren das Projekt in eine IDE. (für dieses Beispiel verwende ich Eclipse IDE 2020).



4. Exportierung des Projekt schließlich als jar-executable Datei (Diese JAR-ausführbare Datei dient zur Benutzerdokumentation).



Die Ausführung des Programms kann auch von EclipseIDE aus erfolgen, wo wir im Folgenden den Java-Code sehen werden, der den wichtigsten Klassen entspricht:

#### Java- und JavaFX-Code:



## Grafische Benutzeroberfläche - Definition von Variablen für Buttons, Scenes, Layouts und Stages

```
3
4 import javafx.application.Application;
5 import javafx.geometry.Pos;
6 import javafx.scene.Scene;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.Label;
9 import javafx.scene.layout.VBox;
10 import javafx.stage.Stage;
11
12 public class GUIMenuDemos extends Application {
13
14
15     Scene sceneMenu, sceneTranscriber, sceneAssistant, sceneSpeaker, sceneAligner;
16     Button btnTranscriber, btnTranscriber1, btnAssistant, btnSpeaker,
17     btnSpeaker1, btnAligner, btnAligner1, btnRec, btnDigit, btnDialog,
18     btnBank, btnBack1, btnBack2, btnBack3, btnExit;
19     Stage alertWindow;
20     Stage windows;
21
22     @Override
23     public void start(Stage primaryStage) throws Exception {
24
25         //Main Menu Container
26         primaryStage.setTitle("GUI Voice Recognition Main Menu");
27
28         Stage Window = new Stage();
29
30         //Scene 1 - Main Menu -----
31         Label labelMenu= new Label("Main Menu");
32         Button btnTranscriber = new Button("Transcriber");
33         Button btnAssistant = new Button("Assistant");
34         Button btnSpeaker= new Button("SpeakerId");
35         Button btnAligner= new Button("Aligner");
36         Button btnExit = new Button("Exit");
37
```

## Grafische Benutzeroberfläche - Definition der Parameter jedes Containers oder spezifischen Layouts entsprechend den Schaltflächen

```
37
38
39
40 btnTranscriber.setOnAction(e -> primaryStage.setScene(sceneTranscriber));
41
42 btnAssistant.setOnAction(e -> primaryStage.setScene(sceneAssistant));
43
44 btnSpeaker.setOnAction(e -> primaryStage.setScene(sceneSpeaker));
45
46 btnAligner.setOnAction(e -> primaryStage.setScene(sceneAligner));
47
48
49 VBox layoutMenu = new VBox(20);
50 layoutMenu.getChildren().addAll(labelMenu, btnTranscriber, btnAssistant, btnSpeaker, btnAligner, btnExit );
51 layoutMenu.setAlignment(Pos.BASELINE_CENTER);
52 sceneMenu = new Scene(layoutMenu, 300, 300);
53
54 //Scene 2 - TranscriberButton -----
55 Label labelTranscriber= new Label("TranscriberDemo Menu");
56
57 //TranscriberDemo
58 Button btnTranscriber1= new Button("TranscriberDemo");
59 btnTranscriber1.setOnAction(e -> {
60     if (! Window.isShowing()) {
61         btnTranscriber1.setOnAction(evn -> Window.setScene(sceneTranscriber));
62
63         try {
64             TranscriberDemo.main(null);
65         } catch (Exception e1) {
66             // TODO Auto-generated catch block
67             e1.printStackTrace();
68         }
69     }
70 });
71
```

## Grafische Benutzeroberfläche - Zurück-Taste, um zur Szene oder zum Hauptmenü zurückzukehren

```
73 //RecDemo
74 Button btnRec= new Button("RecDemo");
75 btnRec.setOnAction(e -> {
76     if (! Window.isShowing()) {
77         btnRec.setOnAction(evn -> Window.setScene(sceneTranscriber));
78         try {
79             //RecDemo.main(null);
80         } catch (Exception e1) {
81             // TODO Auto-generated catch block
82             e1.printStackTrace();
83         }
84     }
85 });
86
87
88 //Back to the main menu
89 Button btnBack1= new Button("Back");
90 btnBack1.setOnAction(e -> primaryStage.setScene(sceneMenu));
91
92 VBox layoutTranscriber= new VBox(20);
93 layoutTranscriber.getChildren().addAll(labelTranscriber, btnTranscriber1, btnRec, btnBack1 );
94 layoutTranscriber.setAlignment(Pos.BASELINE_CENTER);
95 sceneTranscriber= new Scene(layoutTranscriber,300,300);
96
```

## Grafische Benutzeroberfläche - Funktion zum Beenden des Programms und Standardfunktion zum Starten des Programms

```
204
205 //Exit Button action handler
206 btnExit.setOnAction(e -> {
207     if (! Window.isShowing()) {
208         btnExit.setOnAction(evn -> Window.setScene(sceneMenu));
209         System.exit(0);
210     }
211 });
212
213 primaryStage.setScene(sceneMenu);
214
215 primaryStage.show();
216
217 }
218
219
220 public static void main(String[] args) throws Exception {
221
222     //GUI
223     launch();
224
225 }
226
227 }
```

## Rec Demo - Option zum Aufnehmen von Audio im WAV-Format für TranscriberButton - Einstellen des Aufnahmeparameters auf RECORD-TIME, 60 Sekunden

```
3 import javax.sound.sampled.*;
4
5 import javafx.application.Application;
6 import javafx.stage.Stage;
7
8 import java.io.*;
9
10 /**
11  * A sample program is to demonstrate how to record sound in Java
12  */
13 public class RecDemo extends Application {
14     // record duration, in milliseconds
15     static final long RECORD_TIME = 60000; // 1 minute
16
17     // path of the wav file
18     File wavFile = new File("resources/transcriber/RecordAudio.wav");
19
20     // format of audio file
21     AudioFormat.Type fileType = AudioFormat.Type.WAVE;
22
23     // the line from which audio data is captured
24     TargetDataLine line;
25 }
```

## Rec Demo - Definition von Audioformatspezifikation

```
26Ⓢ /**
27  * Defines an audio format
28  */
29Ⓢ AudioFormat getAudioFormat() {
30     float sampleRate = 16000;
31     int sampleSizeInBits = 8;
32     int channels = 2;
33     boolean signed = true;
34     boolean bigEndian = true;
35     AudioFormat format = new AudioFormat(sampleRate, sampleSizeInBits,
36                                         channels, signed, bigEndian);
37     return format;
38 }
39
```

## Rec Demo - Tonaufnahmefunktion im WAV-Format

```
40Ⓢ /**
41  * Captures the sound and record into a WAV file
42  */
43Ⓢ void start() {
44     try {
45         AudioFormat format = getAudioFormat();
46         DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
47
48         // checks if system supports the data line
49         if (!AudioSystem.isLineSupported(info)) {
50             System.out.println("Line not supported");
51             System.exit(0);
52         }
53         line = (TargetDataLine) AudioSystem.getLine(info);
54         line.open(format);
55         line.start(); // start capturing
56
57         System.out.println("Start capturing...");
58
59         AudioInputStream ais = new AudioInputStream(line);
60
61         System.out.println("Start recording...");
62
63         // start recording
64         AudioSystem.write(ais, fileType, wavFile);
65     } catch (Exception e) {
66         e.printStackTrace();
67     }
68 }
```

### Rec Demo - Abschluss des Tonaufnahmeprozesses

```
73  /**
74   * Closes the target data line to finish capturing and recording
75   */
76  void finish() {
77      line.stop();
78      line.close();
79      System.out.println("Finished");
80  }
81
82  /**
83   * Entry to run the program
84   */
85  public static void main(String[] args) {
86      final RecDemo recorder = new RecDemo();
87
88      // creates a new thread that waits for a specified
89      // of time before stopping
90      Thread stopper = new Thread(new Runnable() {
91          public void run() {
92              try {
93                  Thread.sleep(RECORD_TIME);
94              } catch (InterruptedException ex) {
95                  ex.printStackTrace();
96              }
97              recorder.finish();
98          }
99      });
```

### Dialog Demo - Sprachbefehlsoptionen für die AssistantButton-Klasse - Aufruf der Klassen LiveSpeechRecognizer und Configuration

```
4  import java.io.IOException;
5
6
7  import edu.cmu.sphinx.api.Configuration;
8  import edu.cmu.sphinx.api.LiveSpeechRecognizer;
9  import edu.cmu.sphinx.api.SpeechResult;
10
11
12  public class DialogDemo {
13
14      private LiveSpeechRecognizer recognizer;
15
16      boolean active = false;
17
18  public void InitDialog() throws Exception {
19
20      //Configuration Object
21      Configuration configuration = new Configuration();
22
23      // Set path to the acoustic model.
24      configuration.setAcousticModelPath("resources/edu/cmu/sphinx/models/en-us/en-us");
25      // Set path to the language model (vocabulary model).
26      configuration.setDictionaryPath("resources/assistant/2729.dic");
27      configuration.setLanguageModelPath("resources/assistant/2729.lm");
28  }
```



Dialog Demo - Boolean-Aktivierung des Erkennungsprozesses (Wenn der Sprachbefehl Übereinstimmt, wird der Prozess aktiviert)

```

77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
    if (voiceCommand.equalsIgnoreCase("Wake Up")) {
        System.out.println(" ");
        System.out.println("Voice Command is " + voiceCommand);
        System.out.println(" ");
        active=true;
        System.out.println(" ");
        System.out.println(" ");
        System.out.println("          Dialog Assistant Menu");
        System.out.println(" ");
        System.out.println(" ");
        System.out.println("          Say: Open/Close Browser");
        System.out.println("          Say: Open/Close Teams");
        System.out.println("          Say: Open/Close Thunderbird");
        System.out.println("          Say: Open/Close Terminal");
        System.out.println("          Say: Back /(back to the main menu)");
        System.out.println(" ");
        System.out.println(" ");
        System.out.println("          Pls choose one of the options by voice:");
        System.out.println(" ");
        System.out.println(" ");
        System.out.println(" ");
    }

```

Dialog Demo - Oder Mit dem Befehl *Back* kann man zum Hauptmenü zurückkehren

```

99
100
101
102
103
104
105
106
107
108
109
110
111
112
    } else if(voiceCommand.equalsIgnoreCase("Back")) {
        System.out.println(" ");
        System.out.println("Voice Command is " + voiceCommand);
        System.out.println(" ");
        active=false;
        System.out.println(" ");
        System.out.println("Thanks for using the Assistant Demo, now you will return to the main menu.");
        System.out.println(" ");
        System.out.println(" ");
        break;
    }

```

Dialog Demo - Während die Erkennung aktiv ist, können Sie Sprachbefehle verwendet werden , um Aktionen wie das Öffnen oder Schließen des Browsers auszuführen

```

114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
    if(active==true) {
        //Open Browser
        if (voiceCommand.equalsIgnoreCase("open browser")) {
            System.out.println(" ");
            System.out.println("Voice Command is " + voiceCommand);
            System.out.println(" ");
            Runtime.getRuntime().exec("firefox www.google.com");
            System.out.println("The Browser is Opened..");
        } else if (voiceCommand.equalsIgnoreCase("close browser")) {
            System.out.println(" ");
            System.out.println("Voice Command is " + voiceCommand);
            System.out.println(" ");
            Runtime.getRuntime().exec("pkill firefox");
            System.out.println("The Browser is Closed");
        }
    }

```

## Dialog Demo - Hauptmethode zum Starten der Erkennung Startmethoden

```
174
175 //Back to the main menu
176 } else if (voiceCommand.equalsIgnoreCase("back")) {
177     System.out.println(" ");
178     System.out.println("Voice Command is " + voiceCommand);
179     System.out.println(" ");
180     System.out.println("Thanks for using the Assistant Demo, now you will return to the main menu.");
181     break;
182 }
183 }
184 }
185 }
186 }
187
188 public static void main(String[] args) throws Exception {
189     DialogDemo dd = new DialogDemo();
190     dd.InitDialog();
191 }
192 }
193 }
194 }
195 }
```

Sobald die Funktionsweise des Programms überprüft wurde, nachdem die vorherigen Schritte ausgeführt wurden, wird optional die Automatisierung des Maven-Projekts mit *Jenkins* implementiert, sodass wir parallel zur Ausführung des Programms periodische Ergebnisse von Code-Extrakten oder -Funktionalitäten liefern können.

*Bei der Entwicklung und dem Testen dieses Programms wurde ein zuvor konfigurierter Server verwendet. Im Folgenden werden jedoch die Schritte zum Konfigurieren eines lokalen Servers für diesen Fall aufgeführt. Quelle: <https://www.jenkins.io/doc/book/installing/linux/>*

5. Gehen zur Website <https://www.jenkins.io/download/> und herunterladen die Generic-Java-Package (.war) des Programms

### Download Jenkins 2.290 for:

Generic Java package (.war)

SHA-256: 2a81049178368e14a15eacb4b7228bbda0dc42112c2a8d83371aec02ab198707

6. Entpacken den heruntergeladenen Ordner und öffnen das Befehlsterminal

```
bash-4.2$ ls -al jenkins
total 194456
drwxr-xr-x. 2 martin users      71 Jun 16  2020 .
drwxr-xr-x. 3 martin users    21 Apr  9  2020 ..
-rw-r--r--. 1 martin users 66437256 Jun 23  2020 jenkins.war
-rw-r--r--. 1 martin users 66239216 Jun 16  2020 jenkins.war.bak
-rw-r--r--. 1 martin users 66437256 Jun 23  2020 jenkins.war.tmp
```

7. Ausführen den Befehl `java -jar jenkins.war --httpPort=8080` und warten, bis die Seite Jenkins entsperren angezeigt wird.

Getting Started


## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password



Continue

8. Kopieren aus der Jenkins-Konsolenprotokollausgabe das automatisch generierte alphanumerische Kennwort

```
beans [filter, legacy]; root of Factory hierarchy
Sep 30, 2017 7:18:39 AM jenkins.install.SetupWizard init
INFO:
*****
*****
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
2f864d3663814887964b682940572567
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****
*****
---> setting agent port for jnlp
---> setting agent port for jnlp... done
Sep 30, 2017 7:18:51 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Sep 30, 2017 7:18:52 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Sep 30, 2017 7:18:52 AM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
Sep 30, 2017 7:18:52 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
Sep 30, 2017 7:18:59 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
Sep 30, 2017 7:18:59 AM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata 25.543 ms
```

9. Fügen auf der Seite Jenkins entsperren dieses Kennwort in das Feld Administratorkennwort ein und klicken auf Weiter.
10. Nach dem Entsperren von Jenkins wird die Seite Jenkins anpassen angezeigt. Hier können im Rahmen Ihrer Ersteinrichtung eine beliebige Anzahl nützlicher Plugins installieren. Klicken auf eine der beiden angezeigten Optionen:



## Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

### Install suggested plugins

Install plugins the Jenkins community finds most useful.

### Select plugins to install

Select and install plugins most suitable for your needs.

11. Nachdem Jenkins mit Plugins angepasst haben, werden von Jenkins aufgefordert, Ihren ersten Administratorbenutzer zu erstellen.

## Create First Admin User

Usuario:

Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

Wenn die neuen Benutzerdetails eingegeben wurden, wird die Seite "Jenkins ist bereit" angezeigt. Klicken auf "Start using Jenkins"

# Jenkins is ready!


Your Jenkins setup is complete.


[Start using Jenkins](#)


12. In diesem Schritt muss nur noch ein neuer Job erstellt werden, und dafür verwenden wir den Pipeline-Job

**Enter an item name**

» Required field

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

13. Wenn die neue Job-Pipeline erstellt wird, gehen wir zur Konfiguration und geben im Pipeline-Abschnitt den folgenden Code pipeline script type ein:

**Pipeline-Script - Repository-Überprüfung durch Anmeldeinformationen, Verbrauch lokaler Variablen und Ausführung von Maven-Befehlen zur Ausführung des entsprechenden Komponententests**

```
1
2 JDK_PATH = "/usr/java/jdk1.8.0_112/bin"
3 JDK_HOME = "/usr/java/jdk1.8.0_112/"
4
5 // start of the pipeline
6 pipeline {
7
8   agent {
9     label 'master'
10  }
11
12   stages {
13     stage('Checkout the Git Repository and run the internal Junit Tests') {
14       steps {
15         dir('pa-voice') {
16           git branch: 'master',
17             credentialsId: 'ad345c0e-7ea7-4f0c-831a-9dfc9f631c43',
18             url: 'https://gitrepos.aicas.de/blauth/pa-voice.git'
19         }
20         sh "export PATH=${JDK_PATH}:/tools/bin:$PATH && export JAVA_HOME=${JDK_HOME} && cd /home/nanna/.jenkins/"
21       }
22     }
23   }
24 }
25
26 }
```

Wenn auf diesen und den gesamten Quellcode des Programms zugreifen möchten, muss man das offizielle Gitlab-Repository klonen.

Offizielles Repository: <https://gitrepos.aicas.de/blauth/pa-voice.git>

## A.2.2 Entwicklerdokumentation – Fehlerdokumentation

Dieser Auszug aus der Fehlerdokumentation zeigt wie eventuell gefundene Fehler dokumentiert werden.

### **Fehler : MavenProject-Integration in Jenkins**

- **Wie hat sich der Fehler ausgewirkt?**

Der Fehler stellte sicher, dass die Ausführungen von der Jenkins-Seite weiterhin fehlschlagen. Befolgen Sie dazu die Schritte des Herstellers (Jenkins) und die Verwendung des entsprechenden Plugins und der entsprechenden Anmeldeinformationen.

- **Mögliche Folgen des Fehlers?**

Dies kann dazu führen, dass der Entwickler mehrere Fehlermeldungen erhält, die die automatische Testphase blockieren. Darüber hinaus kann dies zu unnötiger Zeitverschwendung führen, da CI nicht integriert wird.

- **Fehlerursache?**

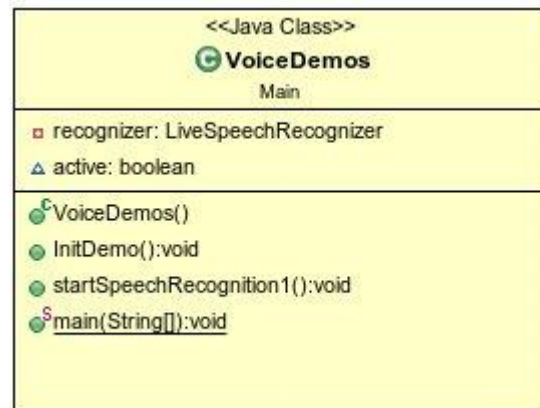
Trotz der korrekten Aktualisierung der Jenkins-Version sowie der Eingabe der Anmeldeinformationen, die dem offiziellen Repository entsprechen, in dem die Tests gehostet wurden. Der Fehler blieb bestehen, da auf dem Jenkins-Server nur eine alte Version von JDK ausgeführt wurde, die einen Kompatibilitätskonflikt verursachte.

- **Wie wurde der Fehler behoben?**

Es wurde ein Pipelinemodul verwendet, und über Code wird der spezifische Pfad der neuesten JDK-Version aufgerufen, der von der Ausführung akzeptiert wird und eine positive Ausgabe generiert

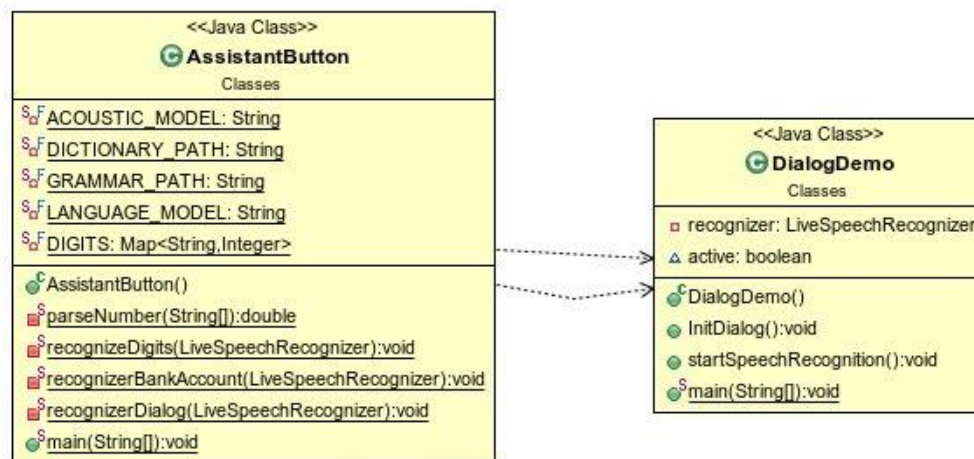
## A.3 UML-Klassen, Flow und Nassi-Diagramm

*UML-Diagramm der Hauptklasse*



In dieser Hauptklasse wird eine Boolesche Funktion verwendet, um das Programm mit dem in der InitDemo-Methode definierten Weck-Sprachbefehl einzuschalten. So werden das Menü und seine verschiedenen Optionen über die IDE-Konsole gedruckt.

*UML-Diagramm - AssistantDemo und DialogDemo*



In diesem UML-Diagramm können wir sehen, dass die AssistantButton-Klasse die Methoden enthält, die RecognizeDigits () und RecognizerBankAccount () entsprechen, die zur Ausführung einfacher mathematischer Funktionen verwendet werden. Als externe Klasse ist die DialogDemo-Klasse jedoch in den Prozess integriert.

### UML-Diagramm - RecDemo



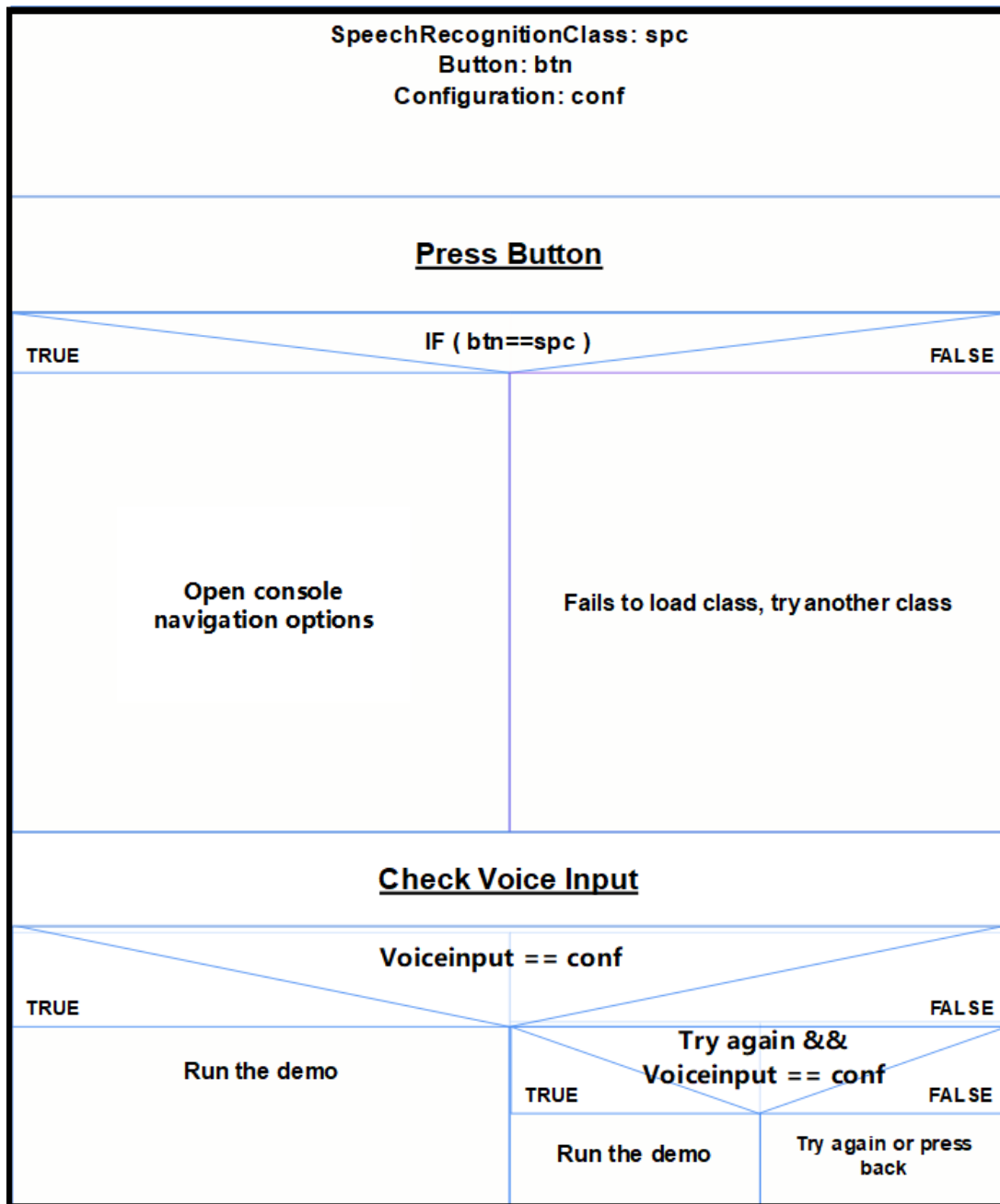
*Diese Klasse wird als optionale Funktion der Transcriber Button-Klasse verwendet, die die Funktion zum Aufzeichnen von Audio für 60 Sekunden bietet. Die abgestufte Oberfläche dieser Funktionalität wurde nicht entwickelt, ist jedoch Teil der anstehenden Punkte, die in Zukunft verbessert werden müssen.*

*Technische Beschreibung: Entwicklung der Mikrofonaufnahmefunktion mittels javax.sound und Entwicklung des Ladens von Audiodateien aus dem Dateisystem (java.io/java.nio) zur Ansteuerung der TranscriberDemo.*

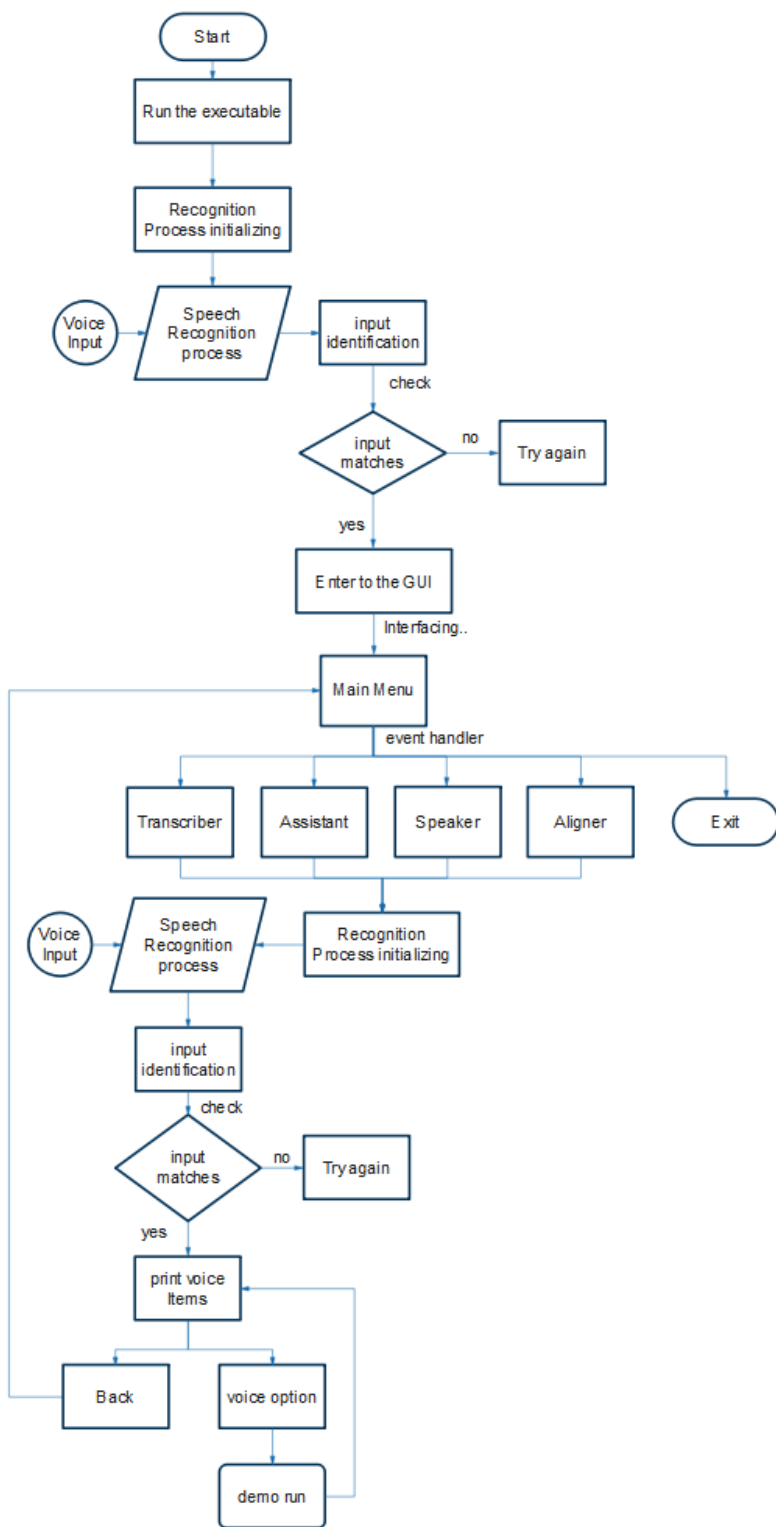
## UML-Diagramm - GUIMenuDemos



In diesem Uml-Diagramm sind die verschiedenen Elemente und Funktionen zu beobachten, die Teil der Entwicklung dieser grafischen Lösung in JavaFX sind.



*Nassi Diagramm - GUI-Nutzungsprozess*



Flow Diagram - Benutzeroberfläche für Spracherkennung mit JavaFX



## A.4 Benutzerdokumentation

Diese Dokumentation enthält die erforderlichen Schritte, um die Demo in ihrer ausführbaren und automatischen Version auszuführen.

In den folgenden Schritten werden die Optionen für Windows, Linux und mit Screenshots beschrieben, die der Benutzer beim Zugriff auf das für diese Demonstration erstellte Programm hat:

### Gebrauchsanweisung für Windows:

1. Klicken Sie mit der rechten Maustaste auf die JAR-Datei.
2. Gehen Sie zu der Option zum Öffnen mit
3. Wählen Sie Java (TM) Platform SE

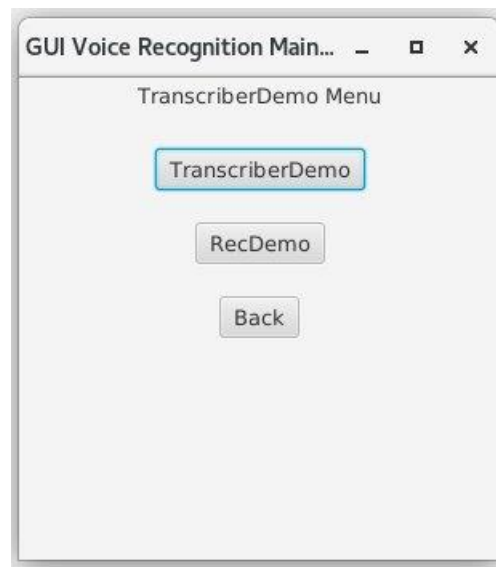
### Gebrauchsanweisung für Linux:

1. Befehlsterminal öffnen
  2. Gehen Sie zum spezifischen Pfad der JAR-Datei
  3. Führen Sie die Befehle `java -jar name_of_the_jar.jar` aus
- Screenshots der Oberflächen:

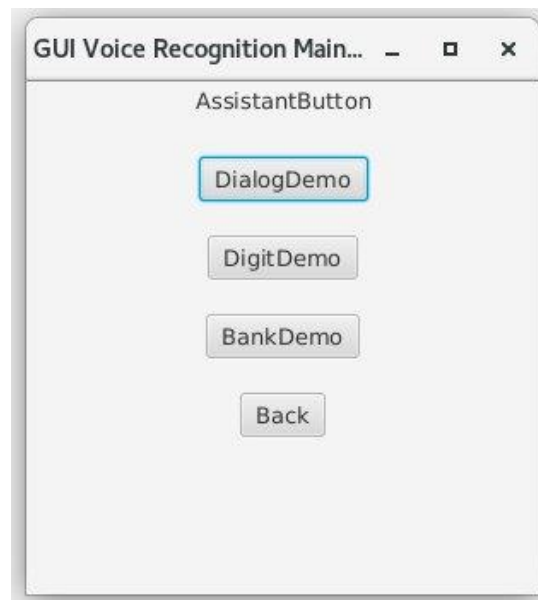
#### Grafische Benutzeroberfläche - Hauptmenü



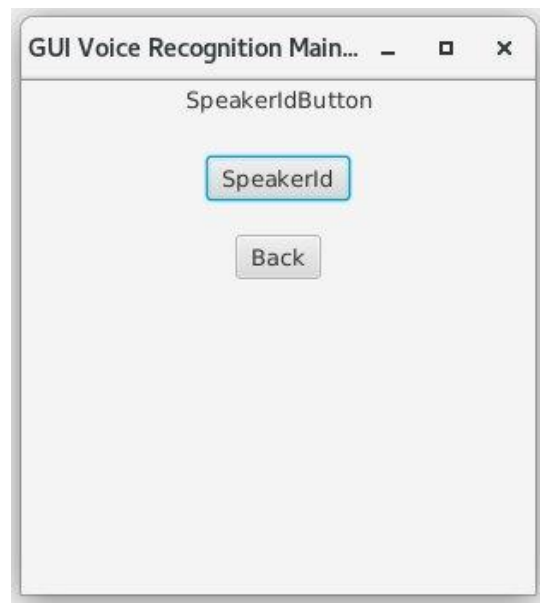
### Grafische Benutzeroberfläche - TranscriberButton



### Grafische Benutzeroberfläche - AssistantButton




### Grafische Benutzeroberfläche - SpeakerIDButton



### Grafische Benutzeroberfläche - AlignerButton



Hinweis: Mit der  Taste endet der Vorgang in der entsprechenden Szene und kehrt zum Hauptmenü zurück

Ausschnitt aus der Benutzerdokumentation - Tastenbedienung

<b>Tasten</b>	<b>Bedeutung global</b>	<b>Bedeutung einzeln</b>
	Die Transcriber-Taste: Ruft die Funktionen TranscriberDemo, RecDemo und Zurück auf	Erkennt und transkribiert den Inhalt einer Audiodatei im WAV-Format
	Die Assistent-Taste: Ruft die Funktionen BankDemo, DigitDemo, DialogDemo und Zurück auf	Ausführung von Assistenzfunktionen mit Sprachbefehlen
	Die SpeakerId-Taste: Ruft die Funktionen SpeakerIdDemo und Zurück auf	Erkennung der Anzahl der Lautsprecher in einer Audiodatei im WAV-Format
	Die Aligner-Taste: Ruft die Funktionen AlignerDemo und Zurück auf	erkennt und transkribiert den Inhalt einer Audiodatei im WAV-Format
	Die Exit-Taste: Ruft die Funktion zum Beenden des Programms auf	Ausrichtung der Synchronisationsparameter einer Audiodatei im WAV-Format