Prof. Dr.-Ing. R. Roosmann, M.Sc. H. Plitzner

# Aufgabenblatt 1

Ziel dieser Aufgaben ist es, dass die Studierenden erlernte fortgeschrittene objektorientierte Programmiertechnik anwenden können.

Abgabe: 27.10.21 (Gr.3) + 08.11.21 (Gr. 1+2); erreichbare Punktzahl: 10; Mindestpunktzahl: 8

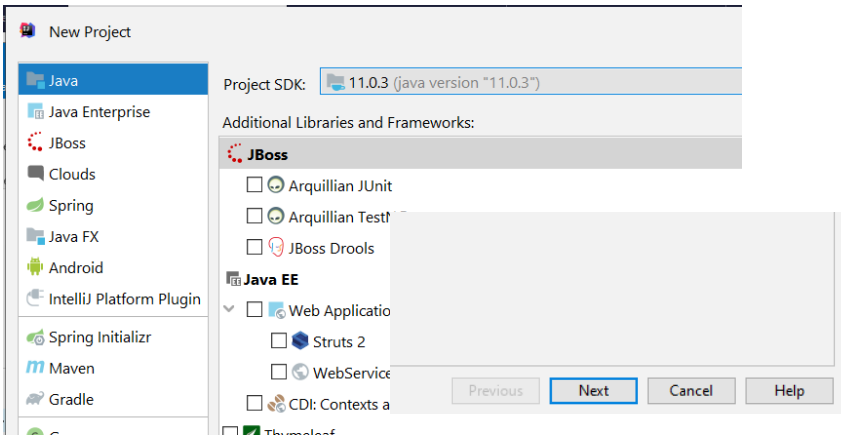
Die Aufgaben sämtlicher Aufgabenblätter sind n Einzelarbeit umzusetzen. Sollte eine Gruppenarbeit erlaubt sein, wird dies bekannt gemacht.

Um Probleme zu besprechen und idealerweise zu lösen, werden Lerngruppen bestehend aus 2-4 Student*innen gebildet. Ziel ist es, Ansprechpartner bei Problemen zu haben. Es ist nicht das Ziel, innerhalb dieser Gruppe Lösungen zu verteilen!! Der Dozent legt die Zusammensetzung der Lerngruppen fest. Nach drei Terminen haben die Studierenden die Möglichkeit, selber Lerngruppen zu bilden. Dazu ist der Dozent spätestens einen Tag vor dem Praktikumstermin über die Zusammensetzung der Gruppe zu informieren.

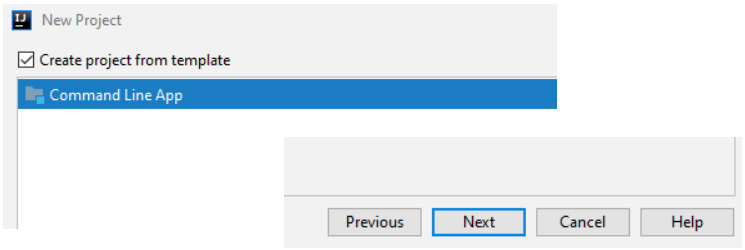
Grundsätzlich: bei Fragen, die nicht im Praktikum geklärt werden können, wenden Sie sich an h.plitzner@hs-osnabrück.de; auch außerhalb der Praktikumszeiten (Termin vereinbaren)

**Aufgabe 1.1: Hallo Nachbar**

Starten Sie IntelliJ und erzeugen ein neues Projekt. Wählen Sie Java und als Project SDK die Version 11.0.x. Ansonsten benötigen wir aktuell nichts weiter und somit klicken Sie anschließend den Next-Button.



Im nächsten Fenster selektieren Sie „Create project from template“ und wählen „Command Line App“.




Abschließend vergeben Sie einen Projektnamen, bspw. `AB01_1`, den Speicherort für Ihr Projekt, bspw. `z:\studium\wise2021\prog3\praktikum\ab1\aufgabe1` und das Base-Package, bspw. `de.hsos.prog3.<ihr_name>.ab01`.

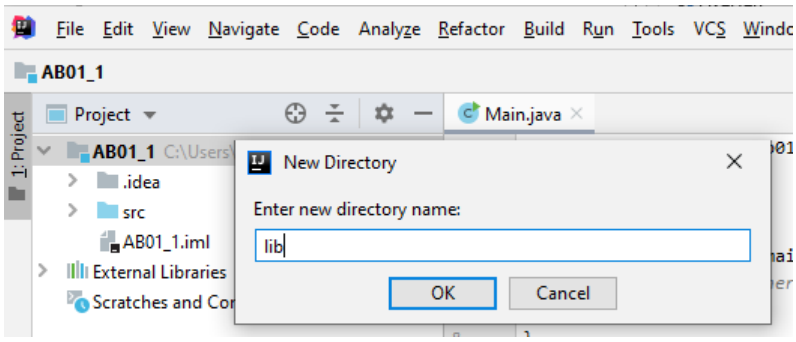
Überprüfen Sie Ihre Wahl des Java Development Kits im nächsten Schritt. Gehen Sie dazu auf File -> Projekt Structure -> Project und sehen Sie sich die Projekt SDK und den Project Language Level an. Gehen Sie dann auf SDKs und sehen sich an, welche SDK auf dem Rechner installiert sind.

Nehmen Sie Kontakt mit 4-8 Teilnehmer*innen der Praktikumsgruppe auf, ermitteln die Vor- und Nachnamen und schreiben ein Programm `Hallo Nachbar`. Schreiben Sie dazu eine Klasse `Nachbar`. Diese sollte zumindest Vor- und Nachname als Instanzvariablen haben. Definieren Sie anschließend eine Klasse App mit einer main-Methode, die diese nutzt. Da Sie mehrere Objekte vom Typ Nachbar verwalten wollen, sollten Sie eine passende Collection auswählen. Vorgabe: stellen Sie sicher, dass jedes Nachbar-Objekt exakt einmal in der Collection vorkommt (Uniqueness). Geben Sie anschließend den Text „Hallo <Vorname> <Nachname>, …, <Vorname> <Nachname>“ aus.

**Aufgabe 1.2 Verwenden einer externen Bibliothek** (`SimpleAudioPlayer.jar`)

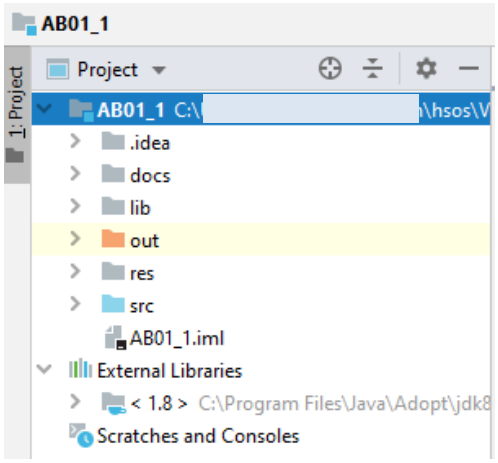
In dieser Aufgabe sollen Sie die externe Bibliothek `SimpleAudioPlayer.jar` verwenden. Die Datei finden Sie in OSCA. Vorgehen zur Einbindung der Bibliothek:

1. Speichern der benötigten Dateien aus OSCA lokal, bspw. unter c:\temp
2. Fügen Sie die Directories *`lib`*, *`docs`* und *`res`* zum Intellij-Projekt hinzu, dies wird für das *`lib`*-Directory im Folgenden beispielhaft beschrieben:
   - Rechtsklick auf das Projekt -> New -> Directory
   - Nennen Sie das Verzeichnis *`lib`* (Sie können aus auch anders benennen)

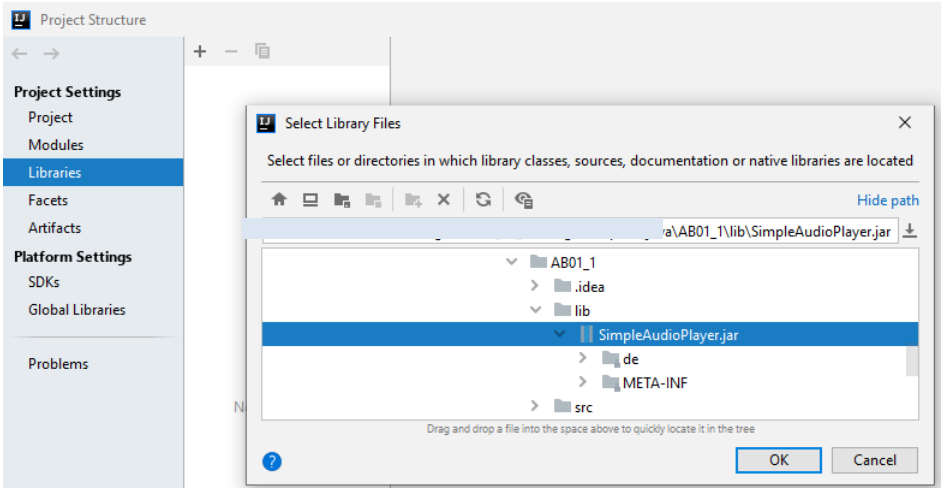



   - Wiederholen Sie das Vorgehen für die Directories *`docs`* und *`res`*

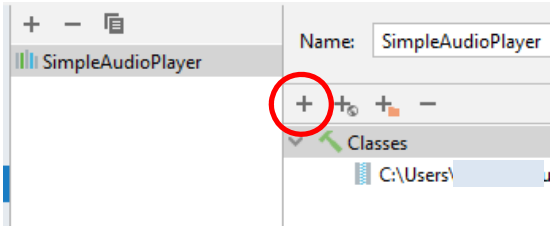
- Ihre Projektstruktur sollte wie folgt aussehen:




3. Kopieren Sie die `SimpleAudioPlayer.jar` im Windows-Explorer mit `Strg+C` und fügen die Datei mit `Strg+V` in Intellij in den *`lib`*-Ordner ein und bestätigen des Kopiervorgangs mit OK. Kopieren Sie auf diesem Wege die `SimpleAudioPlayerDoc.zip` Datei in das *`docs`*- und die wav-Dateien in das *`res`*-Directory.
4. Im nächsten Schritt wird die Bibliothek bekannt gemacht. Dies kann in IntelliJ auf drei unterschiedlichen Ebenen geschehen: a) global für alle Projekte, b) für dieses Projekt und c) für ein Modul. Infos zur Einbindung externer Bibliotheken finden Sie bspw. unter: `https://www.jetbrains.com/help/idea/library.html`
   In diesem Fall soll die Bibliothek für das Projekt wie folgt bekannt gemacht werden:
   - Wählen Sie unter File -> Project Structure -> Libraries
   - Linksklick auf den *`New Library`* Button („+“-Zeichen) -> Auswahl von „Java“ -> `Datei /lib/SimpleAudioPlayer.jar` selektieren und OK klicken:

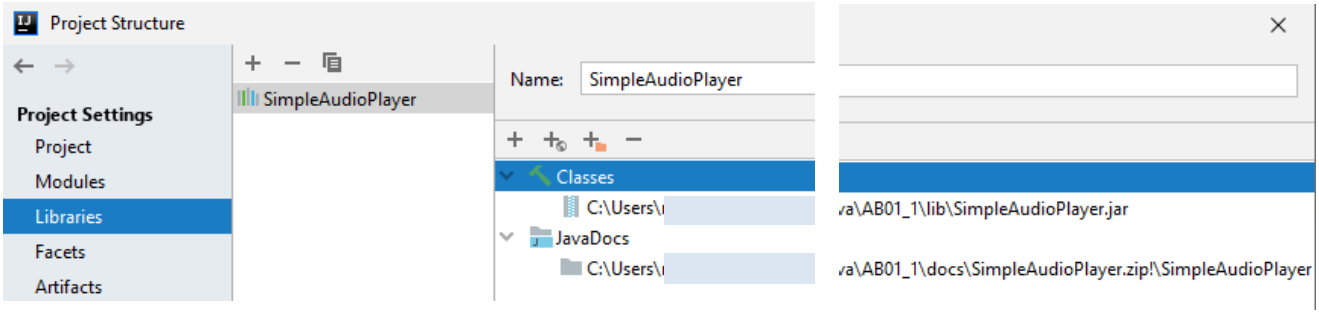



   - Da die gezippten JavaDoc-Dateien zum `SimpleAdioPlayer` vorliegen, ordnen wir diese der `SimpleAudioPlayer`-Library wie folgt zu:



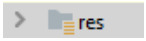

   - Klicken Sie den *Add*-Button, wie in der Abbildung rot umrandet dargestellt

- Definieren Sie den Pfad zur `SimpleAudioPlayerDoc.zip` Datei, wählen dort `SimpleAudioPlayer` und klicken den OK-Button
- Ihre Project-Structure sollte jetzt so aussehen




Damit die JavaDoc-Informationen auch zur Entwicklungszeit über IntelliJ angezeigt werden, bspw. wenn Sie mit der Maus über die Klasse oder eine Methode gehen, wählen Sie unter `File -> Settings -> Editor -> General -> Show quick documentation on mouse move` und klicken Sie dann OK.

5. Java bietet die Möglichkeit, Dateien die zur Ausführung des Programms benötigt werden als Resource bekannt zu machen. Dies sind bspw. Konfigurationsdateien, Images u.a.m. Diese Dateien werden in einem speziellen Resource-Verzeichnis abgelegt. Das Verzeichnis wird beim Build-Prozess bekannt gemacht und im Source-Code lassen sich darüber Resources ohne Angabe des konkreten Pfades zu diesem Resource-Verzeichnis nutzen. Der Zugriff auf Resources erfolgt über den `Classloader`.
   Infos bspw.: https://docs.oracle.com/javase/8/docs/technotes/guides/lang/resources.html

   Damit IntelliJ das *`res`*-Directory als Ressourcen-Verzeichnis erkennt, muss dieses dem Programm bekannt gemacht werden. Führen Sie einen Rechtsklick auf das *`res`*-Directory aus, wählen den Menüpunkt *`Mark Directory As`* und selektieren *`Resources Root`*. Jetzt sollte das *`res`*-Verzeichnis wie folgt aussehen: 


Puh! Nach dieser langen Vorbereitung, steht jetzt die externe Bibliothek SimpleAudioPlayer.jar, inkl. JavaDoc endlich zur Verfügung. Zudem können die Audio-Dateien im wav-Format als Resources verwendet und auf diese Datein unabhängig vom Pfad zum Resources-Verzeichnis über den ClassLoader zugegriffen werden.

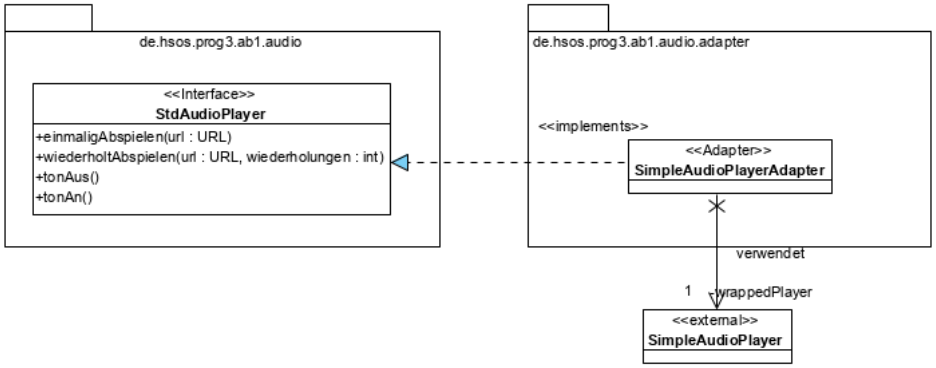
Folgender Source-Code stellt beispielhaft die Verwendung des SimpleAudioPlayer dar. Es ist ein Bariton-Saxophon zu hören (ggf. mit eigenem Kopfhörer in der EDVSZ-Umgebung prüfen;-):

Main.java

```
1  package de.hsos.prog3.ab01;
2
3  import de.hsos.prog3.audio.SimpleAudioPlayer;
4  import java.io.IOException;
5  import java.net.URL;
6
7  public class Main {
8      public static void main(String[] args) throws IOException {
9          URL url = Main.class.getResource( name: "/Baritone.wav");
10         SimpleAudioPlayer player = new SimpleAudioPlayer(url);
11         player.setDebug(false);
12         player.verboseLogging( isVerbose: true);
13         player.play( loop: 0);
14     }
15 }
```

**Aufgabe 1.3: Adapter für den SimpleAudioPlayer**

Wie der Entwickler des SimpleAudioPlayers selber feststellt, wird es in Kürze zu Änderungen der Funktionalität mit Auswirkungen auf die API kommen. Um auf diese Änderungen der externen Bibliothek vorbereitet zu sein, soll das Adapter-Muster umgesetzt werden:




Aufgabe:

1. Definieren Sie das Interface `de.hsos.prog3.ab1.audio.StdAudioPlayer` mit den Methoden (ein Vorschlag; Sie können das Interface auch anders definieren):
   - `void einmaligAbspielen(URL url)`: um die mit dem Übergabeparameter definierte wav-Datei einmalig abzuspielen
   - `void wiederholtAbspielen(URL url, int wiederholungen)`: der Übergabeparameter legt fest, wie häufig der Player die definierte wav-Datei wiederholt abspielt.
   - `tonAus()`: verwendet die Methode `setDebug(true)` und `verboseLogging(true)` des `SimpleAudioPlayers`
   - `tonAn()`: verwendet die Methode `setDebug(false)` und `verboseLogging(true)` des `SimpleAudioPlayers`
   - o

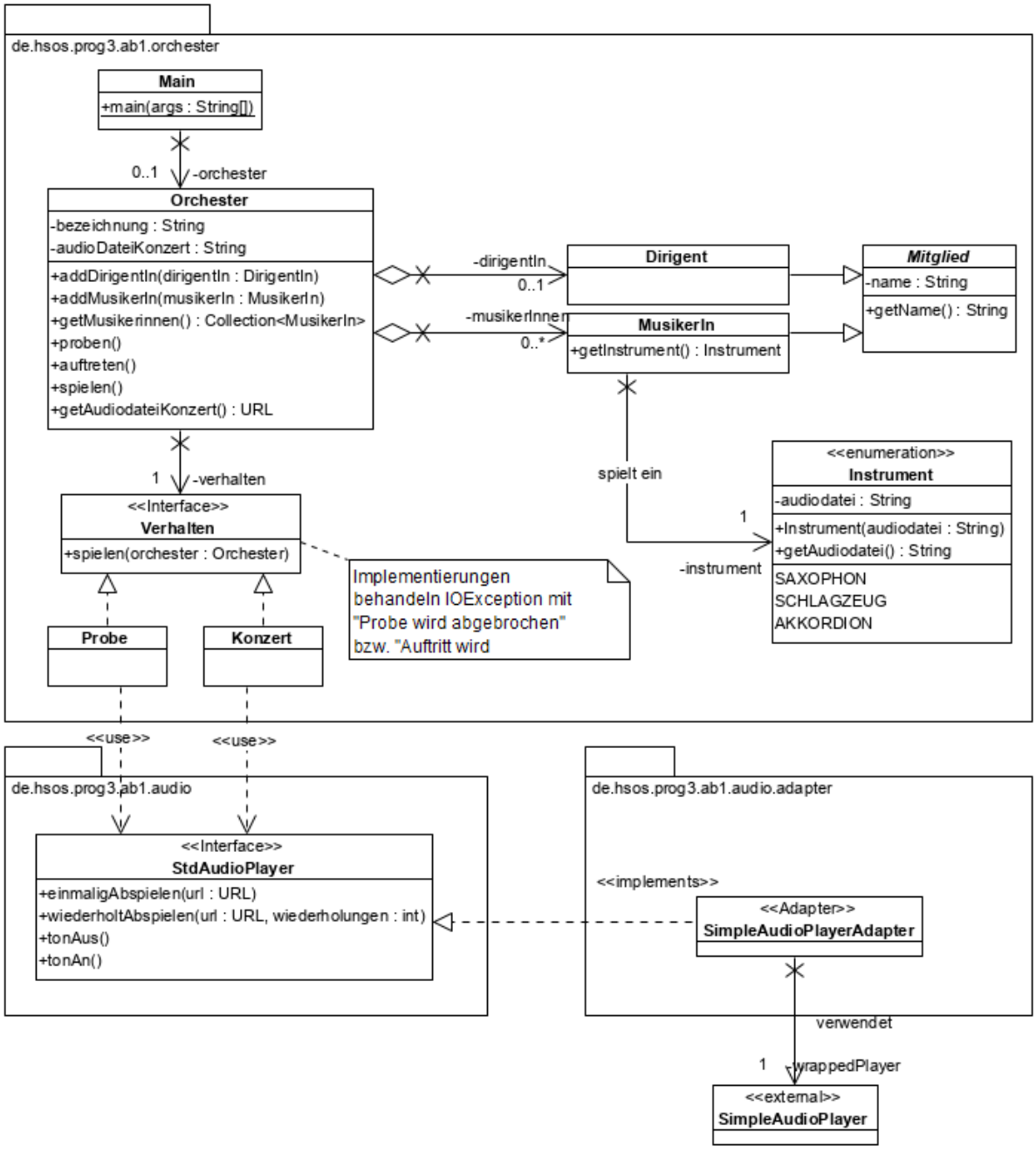
2. Definieren Sie die Klasse de.hsos.prog3.ab1.audio.adapter.SimpleAudioPlayerAdapter, die das Interface StdAudioPlayer implementiert und die Methodenaufrufe an den externen SimpleAudioPlayer weitergibt.

   Zur Information:
   - Der SimpleAudioPlayer kann eine wav-Datei genau einmal abspielen.
   - Die Methoden setDebug(…) und verboseLogging(…) müssen verwendet werden, bevor der SimpleAudioPlayer gestartet wird. Auch wenn die Methodennamen tonAn() und tonAus() suggerieren, dass der Ton ausgestellt werden kann (wenn der Player spielt). In Wirklichkeit ist die Idee, in der Entwicklungsphase den Ton zu deaktivieren und dafür über die Log-Ausgabe anzuzeigen, dass die Datei abgespielt wird (damit die Entwicklerin / der Entwickler weiß, dass die Methode korrekt funktioniert, ohne minutenlang eine wav-Datei abzuspielen).
3. Passen Sie die Main-Klasse aus Aufgabe 1.2 an, so dass die Adapter-Lösung verwendet wird.

**Aufgabe 1.4 Orchester nutzt StdAudioPlayer**

In dieser Aufgabe implementieren Sie ein virtuelles Orchester. Dieses besteht aus einem Dirigenten und mehreren Musiker*innen. Das Orchester hat ein unterschiedliches Verhalten, je nachdem, ob es probt oder einen Auftritt hat. Bei der Probe spielen in unserem Beispiel die Musiker*innen einzeln nacheinander ihr Instrument, bei einem Auftritt, spielt das Orchester zusammen. Um den Sound der Musiker*innen und des Orchesters hörbar zu machen, wird der `StdAudioPlayer` mit dem `SimpleAudioPlayerAdapter` verwendet. Sie simulieren ein Zusammenspiel lediglich und verwenden immer dann, wenn das Orchester spielt eine eigene Datei.

Folgendes UML-Klassendiagramm modelliert das zu implementierende Programm:



Aufgaben:

1. Implementieren Sie einen Aufzählungstyp (`enum`) `Instrument`, mit einer privaten Instanzmethode `audiodatei` vom Typ `String`, einem überladenen Konstruktor und der Methode `String getAudio()`.
   Über diesen Aufzählungstyp sollen die Konstanten SAXOPHON, SCHLAGZEUG und AKKORDION bereitgestellt werden. Weisen Sie diesen Konstanten vom Typ Instrument die unter Aufgabe 1.2 als Java-Resources bereitgestellten wav-Dateien über den Konstruktor sinnvoll zu.
2. Definieren Sie die abstrakte Klasse `Mitglied` mit der privaten Instanzvariablen `name` vom Typ `String` und der zugehörigen `getter`-Methode.
3. Definieren Sie die konkreten Klassen `DirigentIn` und `MusikerIn` als Spezialisierungen der abstrakten Klasse `Mitglied`. Die Klasse `MusikerIn` hat eine zusätzliche private Instanzvariable `instrument` vom Typ Instrument, einen überladenen Konstruktor zur Übergabe des Instruments und eine `getter`-Methode, um das Instrument später abfragen zu können.
4. Definieren Sie das Interface `Verhalten` mit der einzigen abstrakten Methode `void spielen(Orchester orchester)`. Dieses Interface wird eingeführt, um das unterschiedliche Verhalten des Orchesters bei der Probe und beim Auftritt umzusetzen.
5. Definieren Sie die Klasse `Orchester`. Ein Orchesterobjekt hat folgende Instanzvariablen:
   - `String bezeichnung`: Bezeichnung des Orchesters
   - `DirigentIn dirigentIn`: ein Objekt vom Typ DirigentIn
   - `Collection<MusikerIn> musikerInnen`: verwaltet die `MusikerIn`-Objekte des Orchesters. Verwenden Sie einen Collection-Typ, der sicherstellt, dass ein `MusikerIn`-Objekt exakt einmal in der Collection vorkommt.
   - `Verhalten verhalten`: das unterschiedliche Verhalten des Orchesters bei der Probe und beim Konzert soll im Typ `Verhalten` gekapselt werden.
   - `String audioDateiKonzert`: Identifikation der wav-Resource, die beim Konzert gespielt werden soll.

   Der überladene Konstruktor der Klasse `Orchester`:
   - `Orchester(String bezeichnung, String audioDateiKonzert)`: übergeben wird zum Einen die Bezeichnung des Orchesters und zum Anderen die Identifikation der wav-Resource als String Objekt, bspw. `All_Together.wav`

   Die Klasse `Orchester` bietet folgende Methoden an:
   - `public void addDirigentIn(DirigentIn dirigent)`:hinzufügen eines DirigentIn-Objektes
   - `public void addMusikerIn(MusikerIn musikerIn)`: hinzufügen es `MusikerIn`-Objektes
   - `public Collection<MusikerIn> getMusikerInnen()`: `getter`-Methode der Instanzvariable `musikerinnen` liefert eine Referenz auf die Collection der `MusikerIn`-Objekte
   - `public URL getAudiodateiKonzert()`: `getter`-Methode für die Instanzvariable `audioDateiKonzert`
   - `public void proben(Orchester orchester)`: setzt ein Objekt vom Typ `Probe` für die Instanzvariable `verhalten`.

- `public void auftreten(Orchester orchester)`: setzt ein Objekt vom Typ `Auftritt` für die Instanzvariable `verhalten`.
- `public void spielen()`: ruft bei der Instanzvariablen verhalten die Methode spielen auf.

6. Definieren Sie die Klasse `Probe` als Implementierung des Interface `Verhalten`. Die Methode `void spielen(Orchester orchester)` fragt vom Übergabeobjekt `orchester` die MusikerInnen ab, iteriert über die Collection, fragt bei jedem `MusikerIn`-Objekt das `Instrument` mit dem Pfad zur wav-Resource ab, erzeugt ein Objekt vom Typ `StdAudioPlayer` mit der URL der wav-Resource und spielt dieses ab. Die MusikerInnen spielen bei der Probe ihre Instrumente hintereinander. IOExceptions werden hier behandelt. Tritt eine IOException auf, wird der Text „Probe wird abgebrochen“ ausgegeben.
7. Definieren Sie die Klasse `Konzert` als Implementierung des Interface `Verhalten`. Die Methode `void spielen(Orchester orchester)` fragt vom Übergabeobjekt `orchester` mit der Methode `getKonzertSound` die wav-Resource ab, die beim Konzert über den `StdAudioPlayer` abgespielt werden soll. IOExceptions werden hier behandelt. Tritt eine IOExcpetion auf, wird der Text „Auftritt wird abgebrochen“ ausgegeben.
8. Schreiben Sie eine Klasse `App` mit einer `main`-Methode, die ein Orchester mit mehreren MusikerInnen bei der Probe und beim Konzert testet, bspw.:

```
5  public class App {
6      public static void main(String[] args) {
7          String audioDatei = "/All_Together.wav";
8          Orchester orchester = new Orchester( bezeichnung: "HSOS Titty Twister Orchestra", audioDatei);
9
10         Dirigent karajan = new Dirigent( name: "Karjan");
11         orchester.setDirigent(karajan);
12
13         MusikerIn trompete = new MusikerIn( name: "Dirk Die Lunge Mueller", Instrument.TROMPETE);
14         MusikerIn akkordion = new MusikerIn( name: "Akki Taste", Instrument.AKKORDION);
15         MusikerIn drum = new MusikerIn( name: "Das Biest", Instrument.SCHLAGZEUG);
16         orchester.addMusikerIn(trompete);
17         orchester.addMusikerIn(akkordion);
18         orchester.addMusikerIn(drum);
19
20         orchester.proben();
21         orchester.konzert();
22         try {
23             orchester.spielen();
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27     }
28 }
```

**Aufgabe 1.5 Refactoring: Implementieren der Klassen Probe und Konzert als innere Klassen**

Der Vorschlag eines „erfahrenen“ Entwicklers ist, die beiden Klassen `Probe` und `Konzert` in die Klasse `Orchester` als private innere Klassen zu verschieben. Führen Sie anschließend die `main`-Methode erneut aus. Machen Sie sich Gedanken über den vorgeschlagenen Ansatz, konkret über die Vor- und Nachteile und dokumentieren diese stichpunktartig. Denken Sie gerne kritisch über den Ansatz nach und entscheiden, ob Sie diesen Ansatz weiter verfolgen würden oder besser nicht!!

Literatur:


- Ullenboom C.: Java ist auch eine Insel, Galileo Computing, Online: http://openbook.rheinwerk-verlag.de/javainsel/
- Inden M. (2020): Der Weg zum Java Profi, dpunkt.verlag


***Viel Erfolg!!***