Systembeschreibung

Die Startroutine des TDD-Trainers wird in der Klasse *App* ausgeführt. Hierbei wird der sogenannte *Selection-Controller* initialisiert, sowie grundlegende Dinge wie die Fenstergröße, Stage, Scene und Fenstertitel festgelegt. Anschließend wird der Controller in eine LinkedHashMap gehangen um im weiteren Verlauf des Programms über einen zugewiesenen Namen darauf zuzugreifen. Der Selection-Controller, welcher seine Daten aus der SelectionView.fxml bezieht und diese in *createWithName()* lädt, wartet nun auf eine Eingabe des Users.Als Standardpfad wird im Projektverzeichnis nach einem Ordner *tasks* gesucht. Dieser Pfad ist änderbar. Darunter wird in einer Listview der Inhalt, gefiltert nach der Dateiendung .xml, angezeigt. Daneben wird eine Beschreibung der Aufgabe ausgegeben, welche aus der aktuell ausgewählten XML-Datei stammt. Folgende Parameter zur Eingabe sind möglich:

- <**exercises**> Ist das root-Element an dieser Stelle.
- <**exercise**> Hier wird im Attribut der Name der Aufgabe festgelegt.
- <**description**> Legt die anzuzeigende Beschreibung bzw. die Aufgabenstellung fest.
- <**class_name**> Hier wird der Name des zu schreibenden Programms festgelegt. Dieser muss übereinstimmen,mit dem Funktionsnamen im Code, da unter diesem Namen, die Datei später abgespeichert wird!
- <**class_code**> Beinhaltet den Code des Programms, z.B. erste Funktionsskelette oder ähnliches.
- <**test_name**> Agiert analog zum <*class_name*> nur dieses Mal für den Test.
- <**test_code**> Ebenfalls analog zu <*class_code*>.
- <**babysteps_value**> Kann True oder False sein und setzt die Einstellung für das Plugin *Babysteps*.
- <**timetracking_value**> Analog für die Trackingfunktion.

Beim Klick auf Start wird nicht nur die Beschreibung, sondern alle Informationen eingelesen und in einem Objekt *Exercise* abgespeichert. Anschließend wird nachgesehen ob eventuell bereits alte Projekte zur Aufgabenstellung existieren. Dies richtet sich nach der Logik unseres Abspeicherungsprinzips:

- Für ein Projekt wird beim ausführen der Funktion *save()* im Aufgabenverzeichnis nachgesehen ob ein Ordner mit dem Namen der Exercise bereits existiert. Wenn nicht wird dieser angelegt und der Code sowie der Test in einer Datei

  <*class_name*>*.java* bzw. <*code_name*>*.java* gespeichert.Sollten diese Dateien bereits existieren werden sie überschrieben!

Also wird der jeweilige Ordner im Aufgabenverzeichnis gesucht und bei Existenz der Dateien, der Inhalt dieser anstatt dem der XML-Datei verwendet. Anschließend wird ein Trainer-Controller erstellt und geladen. Der Trainercontroller, liest die bereits vorhandenen Daten aus dem Objekt *Exercise* aus und gibt Sie an den dafür zuständigen untergeordneten Controller weiter. Hierbei gibt es folgende Möglichkeiten:

- **Description-Controller:** Der *Description-Controller* ist für das Anzeigen der Aufgabenstellung zuständig, welche er aus der *Exercise* ausliest.
- **ErrorAndFailure-Controller:** Hier wird die Anzeige auftretender Test- oder Compilerfehler in den jeweiligen Textfeldern koordiniert.
- **Solution-Controller:** Der *Solution-Controller* regelt, die Deaktivierung des Test- bzw. Codeeditors bzw. die Aktivierung des anderen, das Einlesen der Daten aus der Exercise, sowie die Übergabe des editierten Codes und Testcodes, welchen der *Trainer-Controller* dem Compiler zur Verfügung stellt.

Keine Plugins

Man befindet sich anfangs in der Test-Edit-Phase, bei welcher man versucht einen Testfehler oder Kompilierungsfehler zu provozieren. Wählt man nun *Compile&Run* werden Test und Code, in einem *Compilation*-Objekt gekapselt, an den bereits vorhandenen Compiler aus der Aufgabenstellung übergeben und auf auftretende Fehler überprüft. Sollte es zu einem Fehler kommen wird die Statusleuchte auf rot gesetzt und man erhält die Möglichkeit seinen Code anzupassen.

Wichtig an dieser Stelle ist, dass die Statusleuchte nicht die etwas uneindeutigen Phasen der Aufgabenstellung darstellt, sondern sich nach dem in der Aufgabe verlinkten Artikel von Frank Westphal orientiert

Die vom Compiler ausgegeben Fehler werden an den *ErrorAndFailure-Controller* übergeben, welcher die entsprechenden Textfelder setzt.

Anschließend kann der User den Code editieren und *Compile&Run* oder *EditTest* wählen. *EditTest* lädt an dieser Stelle in das Editierfeld des Codes, den vor dem Eintritt in die *EditTCodePhase* noch im *Solution-Controller* vorhandenen String ohne eventuell erfolgte Änderungen.

In der *Refactor-Phase* läuft alles ähnlich zu der *EditCode*, jedoch mit strengeren Bedingungen, um weiterzukommen.
Bei *exit* wird ein Bestätigungsdialog aufgerufen um dem Nutzer nochmal daran zu erinnern vorher zu speichern. Beim Drücken des *Ja-Buttons* werden alle temporären Variablen wie z.B. *isBabysteps*, *isTracking* etc. auf false, sowie der Timer zurückgesetzt und anschließend der *Selection-Controller* aufgerufen.

Babysteps

Sollte die Funktion *Babysteps* eingeschaltet sein, wird vom Trainer eine Festlegung der Zeitspanne erwartet, mit dessen Setzen der Countdown beginnt runterzulaufen. Dies wird in Abhängigkeit von dem Boolean *isBabysteps* über die Funktion *startTimer()* geregelt. Diese setzt in einem Thread den Timer und regelt abhängig vom aktuellen Zustand ,am Anfang alle Textfelder deaktiviert, bis eine Zeitspanne ausgewählt wird, das weitere Aktivieren/Deaktivieren der Textfelder über die Funktionen *editTest()* und *editCode()*. Diese setzen in Abhängigkeit von *isBabysteps* in jeder Phase den Timer zurück.

Tracking

Beim setzen des Booleans *isTracking* auf True wird ähnlich zu der Zeitmessung im Compiler mithilfe von zwei Long-Variablen die verbrachte Zeit in den Phasen Testedit, Codeedit und Refactor in drei Listen jeweils festgehalten. Analog die jeweils entstandenen Compiler- & Testfehler. Abhängig von *isTracking* wird bei Betätigung des *Exit-Buttons* mit der Funktion *quit()* der *Result-Controller* angelegt, die Zeiten pro Phase aufsummiert und in die BarChart eingetragen, sowie alle Fehlermeldungen in einen jeweiligen ListView eingetragen.