Systembeschreibung zum Projekt7: TestDrivenDevelopment(TDD)

Unser Programm besteht aus 4 Scenes:

- Der Hauptscene mit dem eigentlichen Programm
- Der Analyse Scene für die Analyse des Timetracking Features
- Der Katalogauswahl Scene um eine Aufgabe auszuwählen
- Der Directorychooser Scene

Das Programm beginnt mit der Directorychooser Scene zur Auswahl des Verzeichnisses in welchem die Kataloge hinterlegt werden. Nach der Auswahl werden diese in der Klasse KatalogCreator.java eingelesen. Dabei haben wir uns für das .txt Format entschieden nach folgendem Layout:

AufgabenName:

Beschreibung:

Classname:

ClassHeader:

Testname:

TestHeader:

BabySteps:
true/false
TimeForBabystepps:
Minuten:Sekunden (kann auch leer gelassen werden falls Babystepps false sind)
Timetracking:
true/false

Die einzelnen Dateien werden mit Files.readAllLines zu einer ArrayList gemacht und aus dieser werden die einzelnen Attribute ausgelesen. Daraus wird dann ein Array aus Katalogen erstellt, in dem die jeweiligen Wert gespeichert werden zur späteren Verwendung. In der Funktion createButtonArr wird dann aus dem Katalog Array ein Button Array erstellt mit dem richtigen Format für die Buttons und der setOnAction-Funktion. Schließlich wird das ButtonArray returned und in eine GridPane eingebunden mit dem die Katalogauswahl Scene

erstellt wird. In dem setOnAction Event der Buttons wird eine neue Scene erstellt, nämlich die für das Hauptprogramm, außerdem wird in der static variabel choosenKatalog der ausgewählte Katalog, des dementsprechenden Buttons gespeichert für die spätere Verwendung.
Nach Klicken des Buttons befindet sich der Nutzer im Hauptprogramm, wo zunächst die Controller.initialize Funktion aufgerufen wird um die Labels und TextAreas an ihre Integer/String Properties zu binden. Dann muss der Nutzer den start Button auswählen wodurch die setStart Methode aufgerufen wird. Der Timer wird dann mit resetTimer gestartet und die TextAreas erhalten ihre vorgegeben Test und Code Header. Falls das timetracking Feature true ist für diesen Katalog wird auch dieses gestartet.
ResetTimer() ist eine Funktion zum erstellen oder reseten des Timers. Sie rechnet die secondsForBabystepps aus dem choosenKatalog in die richtigen Werte um, um damit die createTimer Function aufzurufen. Die create Timer Funktion setzt lediglich ein paar Variabeln neu bzw. wieder zurück. Das eigentliche erstellen des Timers erfolgt in der newTimer Funktion, welche einen Timer erzeugt der, bei Zeit angaben > 1 Minute, sich nach 60 Sekunden wieder selbst neu erzeugt und den Propertywert für die Minute um 1 erhöht. Die counter Variabel dient hierbei dazu dafür zu sorgen, dass der Timer nach der richtigen Anzahl Minuten aufhört. Falls man einen Katalog auswählt der ohne Babysteppfeature läuft, wird timeLimitMinutes auf Integer.MAX_VALUE; gesetzt damit er (fast) endlos weiter läuft.
Der Timer ist in der Lage jede Zeit richtig auszuführen, also auch so etwas wie 00:15 oder 2:13.
Sollte er (bei eingeschalteten Babystepps) ablaufen, wird jumpOneStepBack aufgerufen, um eine TDD Phase zurück zu springen. (Auch hier wird dann der Timer resetet)

Nun weiter zum aufruf der setStart Funktion des Start-Buttons:
Hier wird nun Excercise.start() ausgeführt. In der Exercise Klasse wird der choosenKatalog ausgelesen um z.B die ClassHeader oder TestHeader zu erstellen.

Falls nun der Nutzer den NextStepp Button anklickt, wird die Funktion setNextStep aufgerufen.
Hier wird zu allererst der neue Code zum compilen in NextSteper.compileTestGenerator aufgerufen. (Dazu gleich mehr) Nun werden die Ergebnisse des compile Versuchs ausgewertet. Bei einem CompileError wird dieser im RückmeldungsLabel angezeigt. Ebenso wenn es fehlgeschlagene Tests außerhalb der TestschreibenPhase gibt. Innerhalb der Testschreiben Phase ist es wiederrum entscheidend wie viele fehlschlagen. Bei mehr als einem wird der Nutzer darauf hingewiesen das nur ein Test fehlschlagen darf. Bei genau

einem wird er in die nächste Phase weitergeleitet mit allen dazu gehörigen Funktionen(neuen Timer erstellen, TextAreas neu Texte zuweisen, Alert anzeigen, Grafik rotieren lassen).
Bei keinem fehlgeschlagen Test in der WriteTestPhase wird der Nutzer aufgefordert einen fehlschlagenden Test zu schreiben.
Wenn keiner dieser Bedingungen zutrifft handelt es sich um Code aus der Refactor oder Code schreiben Phase der erfolgreich compilt wurde und alle Tests erfüllt und somit wird auch hier der Nutzer in die nächste Phase weitergeleitet.
Zuletzt werden noch die Daten für das Tracking abgespeichert.

Nun zur Klasse NextSteper:
In der Funktion compileTestGenerator wird der Code aus den TextAreas compilt mithilfe der virtual-kata-lib. Es werden 2 Instanzen der Klasse CodeFailure für die spätere Rückgabe erstellt, eine für Code und eine für Test Probleme. Es werden zwei CompilationUnits an die CompilerFactory übergeben um einen JavaStringCompiler zu erhalten, auf dem compileAndRunTests() aufgerufen wird.
Entweder kommt es dann zu compile Erros welche in der CodeFailure Klasse compileFailure gespeichert werden oder es gibt keine Erros, dann muss geguckt werden ob es noch FailedTest gab, welche dann zum testFailure Objekt hinzugefügt werden, falls vorhanden.

Das Tracking wird über die Klassen Tracker und TrackStep realisiert.
In der Tracker.startTracker Funktion wird der Tracker initialisiert mit einem startDate als Anfangszeit und einer ArrayList zum erstellen der Ausgabe. Die Trackingausgabe erfolgt in einer Tracked.txt Datei die im ProjektOrdner liegt. Jedes mal wenn setNextStep aufgerufen wird, wird writeStep aufgerufen. Hier wird ein TrackStep Objekt in der generateStep Funktion erzeugt und an trackWriter übergeben um es in die File zu speichern. GenerateStep() erzeugt dann eine neue LocalDateTime Variabel und berechnet die Differenz zur vorherigen Zeit, die immer in startTime gespeichert ist, aus. Außerdem wird der Inhalt der Lables für Rückmeldung und aktuellerPhase, sowie der Inhalt der TextArea zum Code schreiben an den Constructor von TrackStep übergeben. TrackStep speichert diese Information dann in Instanz Variablen ab, welche in TrackStep.asStringArrayList verwendet werden um die Ausgabe für den User zu erzeugen. Diese erzeugte ArrayList wird dann in Tracker.trackWriter in die Datei geschrieben.

Wenn TimeTracking aktiv ist kann der User den Analyse Button drücken um eine neue Scene mit einer Grafik zur Analyse des Zeit Verbrauchs öffnen. Das Diagramm wird in setOpenAnalyser erstellt. Dazu werden aus den Tracking

Daten die insgesamt verstrichene Zeit für die Test-, Code-, und Refactorphase in einem PieChart Objekt gespeichert. Jeder Abschnitt erhält seinen eigenen CSS Style und einen EventHandler der dafür sorgt das man, wenn man mit der Maus über die einzelnen Abschnitte fährt die verstrichene Zeit in Sekunden angezeigt bekommt.

Zuletzt gibt es noch einen Fullscreen Button, zum wechseln in oder aus dem Fullscreenmode.