

Systembeschreibung zum Projekt7: TestDrivenDevelopment(TDD)

Unser Programm besteht aus vier Scenes:

- Der Hauptszene 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 Werte zur späteren Verwendung gespeichert werden. 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-Variable `choosenKatalog` der ausgewählte Katalog, des dementsprechenden Buttons gespeichert für die spätere Verwendung.

Nach Klicken des Buttons befindet sich der Nutzer im Hauptprogramm, wo zunächst `Controller.initialize()` 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 Reset des Timers. Sie rechnet die `secondsForBabysteps` aus dem `chooseKatalog` in die richtigen Werte um, um damit die `createTimer()` Funktion aufzurufen. `createTimer()` setzt lediglich ein paar Variablen neu bzw. wieder zurück. Das eigentliche erstellen des Timers erfolgt in `newTimer()`, wo ein Timer erzeugt wird, der sich bei Zeitangaben ≥ 1 Minute nach 60 Sekunden wieder selbst neu erzeugt und den Propertywert für die Minute um 1 erhöht. Die `counter` Variable dient hierbei dazu dafür zu sorgen, dass der Timer nach der richtigen Anzahl Minuten aufhört. Falls man einen Katalog auswählt der ohne Babystepfeature 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 Babysteps) ablaufen, wird `jumpOneStepBack()` aufgerufen, um eine TDD Phase zurück zu springen (Auch hier wird dann der Timer zurückgesetzt).

Nun weiter zum Aufruf der `setStart()` Funktion des Start-Buttons: Hier wird nun `Exercise.start()` ausgeführt. In der Exercise Klasse wird der `chooseKatalog` 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 compilieren in `NextStepper.compileTestGenerator` aufgerufen (dazu gleich mehr). Nun werden die Ergebnisse des Compiler-Versuchs ausgewertet. Bei einem `CompileError` wird dieser im Rückmeldungs-Label angezeigt. Ebenso wenn es fehlschlagene Tests außerhalb der Test-Schreiben-Phase gibt. Innerhalb der Test-Schreiben-Phase ist es wiederum 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 nicht fehlschlagendem Test in der Test-Schreiben-Phase wird der Nutzer aufgefordert einen solchen zu verfassen. Wenn keine dieser Bedingungen zutrifft, handelt es sich um Code aus der Refactor- oder Code-Schreiben-Phase der erfolgreich compiliert wurde und alle Tests erfüllt; somit wird auch hier der Nutzer in die nächste Phase weitergeleitet. Zuletzt werden noch die Daten für das Tracking abgespeichert.

Nun zur Klasse `NextStepper`: In der Funktion `compileTestGenerator()` wird der Code aus den Text-Areas mithilfe der `virtual-kata-lib` compiliert. Es werden zwei 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 Compiler-Errors welche in der `CodeFailure` Klasse `compileFailure` gespeichert werden oder es gibt keine Errors, 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 Projekt-Ordner 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` Variable und berechnet die Differenz zur vorherigen Zeit, die immer in `startTime` gespeichert ist, aus. Außerdem wird der Inhalt der Labels für Rückmeldung und aktuelle Phase, sowie der Inhalt der `TextArea` zum Code schreiben an den Constructor von `TrackStep` übergeben. `TrackStep` speichert diese Information dann in Instanzvariablen 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 Zeitverbrauchs öffnen. Das Diagramm wird in `setOpenAnalyzer` 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, dass 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 Wechsel in oder aus dem Fullscreenmode.