Systembeschreibung

Uns ist nicht ganz klar wie das Abgabeformat sein soll. Die Travis Anzeige in GitHub meldet „passing“, von daher gehen wir aus, dass das Programm ordentlich und richtig läuft. Ob nun eine .jar gepusht werden soll oder ähnliches geht aus der Aufgabenstellung nicht hervor, daher belassen wir es bei diesem Stand des Repositories.

Get- und Set-Methoden werden im Folgenden nicht beschrieben, da sie selbsterklärend sind.

Package FileHandling:

Public class Code:

Public Code(String className, String code):

Das Objekt enthält den Klassennamen, den Klasseninhalt und erzeugt dadurch eine CompilationUnit(mit isTest=false) und bestimmt einen boolean ob diese kompilierbar ist.

Public boolean testingCompilationUnit():

Mit Hilfe der vk-Bibliothek wird hier getestet, ob die im Objekt erzeugt CompilationUnit kompiliert. Ist dies nicht der Fall wird eine Fehlermeldung in der Konsole ausgegeben und die Methode gibt false zurück. Andernfalls gibt sie true zurück.

Public void save():

Der aktuelle Inhalt des Code Objekts wird in eine .java Datei mit dem zugehörigen Klassennamen gespeichert.

Public class CodeList:

Public CodeList():

Erzeugt eine leere ArrayList von Code Objekten.

Public boolean compilable():

Die Liste wird zu einem Array umgewandelt. Es wird durch jedes CodeElement iteriert und getestet ob es kompiliert. Sobald eins nicht kompiliert wird die Methode mit dem Rückgabewert false beendet.

Public void saveCode():

Speichert das erste Element der CodeList(ruft die save Methode der Code Klasse auf).

Public class Test:

Public Test(String testName, String TestCode):

siehe public Code(...). Ausnahme isTest = true in der CompilationUnit.

Public boolean oneTestFailing():

Kompiliert und lässt alle Tests laufen, die in dem übergebenen Code enthalten sind.

Dabei wird die Anzahl der fehlgeschlagenen Tests gezählt.

An dieser Stelle ist anzumerken, dass wir uns dafür entschieden haben, dass alle Tests in einer .java Datei zu stehen haben. Ist diese Anzahl gleich eins, gibt die Methode true zurück. Dies ist wichtig für den Übergang von RED zu GREEN.(write one failing test).

Public boolean allTestsPassing():

Analog zu oneTestFailing, außer dass bei einer Anzahl von fails == 0 true zurückgegeben wird. Dies ist notwendig für den Übergang von GREEN zu REFACTOR und REFACTOR zu RED.

Public boolean save():

analog zur save Methode in class Code.

Public class Babysteps:

Public Babysteps(boolean baby, long limit):

Erstellt ein Objekt, wobei limit bestimmt wie lang das Bearbeitungsintervall ist (z.B. 2minuten). Diese Zeit muss als long-Wert (Sekunden) vom Aufgabensteller in der xml-Datei eingetragen werden.

Public class Exercise:

Public Exercise(String description, CodeList klassen, Test test, Babysteps baby, boolean timer):

In dem Objekt steht description für den Text der im Katalog angezeigt wird. Der boolean timer setzt fest ob TimeTracking aktiviert ist.

An dieser Stelle ist anzumerken, dass Babysteps und TimeTracking gleichzeitig aktiviert sein können.

Außerdem wird TimeTracking wie folgt interpretiert: Es wird die Zeit gestoppt die der User pro Phase benötigt. Am Ende wird dies in einem Kuchendiagramm dargestellt.

Public Exercise():

Erzeugt ein default Objekt das beim Start der Anwendung geladen wird.

Public StackPane display():

Implementiert das StackPane, welches den Katalog in einem extra Fenster in einer Art Karteikarten darstellt.

Public class Loader:

Public static void loadExcercise(Exercise exer):

Öffnet das Fenster in dem der User die Katalog.xml Datei auswählen kann.

Private static void laden(String filename, Label anzeige, Stage stage, Exercise exer):

Liest die ausgewählte xml aus und gibt ggf. passende Fehlermeldungen.

private static void katalogView(Document xmldoc,Stage stage, Exercise exer):

stellt die einzelnen Karteikarten, passend zur zuvor ausgelesenen xml Datei, dar und sorgt dafür, dass diese auswählbar(„anklickbar“) sind.

Private static Document loadDoc(String file):

Lädt das angegebene Dokument.

Private static ArrayList<Exercise> parseExercise(Cocument xmldoc):

Liest die xml Datei aus und erstellt eine Liste aus Exercise Objekten mit dem passenden Inhalt pro Aufgabe.

Public class SaveTest & public class LoadTest:

Testet die save bzw load Methoden.

Package tests:

Public class TestMain:

Testet compile-Methoden von Code und CodeList.

An dieser Stelle ist anzumerken, dass für alle anderen Methoden(package logik) keine Tests geschrieben wurden, da diese sozusagen visuell über die GUI von uns getestet werden. Reagieren die Buttons zum Beispiel nicht, funktioniert die Methode nicht.

Package application.logik:

Public class Logik:

Kümmert sich um die Umsetzung von TimeTracking und Babysteps. Erstellt passendes Chart.

Kümmert sich um das zeitlich richtig Aufrufen der Methoden(compile etc).

Public Logik(Exercise e):

Setzt erstmal alle Zeiten für die Tracking-Erweiterungen auf null und speichert die übergebene Exercise.

Package GUI:

Public class Ampel:

Enthält Methoden für den Farb- und Labelwechsel, je nachdem in welcher Phase man ist und welche Erweiterung angeschaltet ist.

Public Ampel():

Erstellt eine Ampel und Statusanzeigen mit Hilfe von Stack-, Grid- und BorderPane.

Die Idee ein GridPane über das StackPane des Ampelgrundes zu legen haben wir von https://kjswebdevelopment.wordpress.com/2014/06/12/simple-java-traffic-light/ (auch wenn es vermuten lässt, dass wir noch mehr von der Seite haben, ist das nicht der Fall)

Public class Controller:

Verwaltet die Grafik, d.h. Die ButtonActions der fxml Buttons werden gesetzt. Mit anderen Worte: Die Grafik wird mit der Logik verknüpft.Public class GUIMain:

Startet die Grafik via fxml.