Package: tddt.UI

Test_UI.java

public static void runTestUI(Exercise exercise):
initiiert die Oberflächel, mit dem der Benutzer sein Programm schreiben und testen kann (TDD)

public static void switchStatus():
setzt den Timer(für das Programmieren) zurück und wechselt den Status des Programms, welcher besagt, in welchem Schritt des TDD der Benutzer sich befindet

public static void returnToTest():
setzt sourceCode zurück auf den Backup der bei jedem switchStatus() initiiert wird und wechselt den Status.

public static void reset():
setzt den Code der grade bearbeitet wird und den Timer zurück

UIRunner.java

public void start(Stage primaryStage):
Initiiert die Stage für das Haputmenü und ruft Loader.ask(...) auf um die Stage koplett, mit den Auswahlmöglichkeiten der Übungen zu befüllen

public void design(Button button, Blend blend):
rein optische Methode. Zuweisung von Size, Effect und Style der Projektnamen.

public void decorate(BorderPane border, Button Neues_Projekt, Button Laden, Button Beenden, Blend blend):
initialisiert das Gridpane zum Laden eines Projekts

public void shadowAndInner(Blend blend):
rein optische Methode. Macht Schrift hübsch

public List<Exercise> readExercise():
Herzstück des Codes.
Initiiert XML Dateien, welche zum compilen und testen des Benutzercodes gebraucht werden. Diese werden mit ihren Argumenten(className, classCode ….) in eine ArrayList eingespeichert. Die ArrayList wird zurückgegeben.

Package: tddt.code

Compile.java

public static void compile(String code,String codeClassName, String tests, String testClassName, TextArea output):
Kompiliert den Benutzercode(source und test) und gibt dem Benutzer CompileErrors zurück.

public static void runTests(String code,String codeClassName, String tests, String testClassName, TextArea output):
Anwendung der Benutzertests auf den Benutzercode. Falls Die Vorraussetzungen für einen Statuswechsel vorhanden sind (bei writeTest 1 Fehler, bei fixTest 0 Fehler) wird switchStatus aufgerufen.

Loader.java

public static void ask(List<Exercise> exercises):
liest aus der ArrayList Exercise bereits vorhandene Projekte aus und koppelt diese an Knöpfe im Hauptmenü, mit denen der Benutzer diese Aufrufen kann.

Timer.java

public Timer(Text text, boolean babystep, int maxSeconds):
initiiert den Timer, welcher benutzt wird um die Zeit zu messen, welche der Benutzer braucht um jeweils den fixText/writeTest code zu schreiben.

public void start():
startet den Timer indem clocking auf true gesetzt wird

public void stop():
stoppt den Timer indem clocking auf false gesetzt wird

public void kill():
killt den Timer indem clocking und running auf false gesetzt werden

public int getTotalSeconds():
Getter: gibt die int seconds am Ende einer Codinginstanz(writeTests/fixTests) zurück

public void reset():
setzt die Int seconds auf 0 zurück

private int getSeconds()/private int getMinutes():
Getter: gibt die Minuten/Sekunden zurück und übersetzt 60 Sekunden in eine Minute

private String MinutesSeconds():
Parsed seconds und minutes in einen String der dem Benutzer angezeigt werden kann.

Tracker.java

public static void startTimer():
initiiert Timer (siehe Timer.start)

public static void displayTimes():
gib bei Aufruf zurück wie lange das schreiben des Codes/Tests gedauert hat

public static void switchStatus():
ändert den Status in Tracker zwischen “code” und “test” Das wird gebraucht um beide Zeiten zu speichern damit man diese am ende eines Programmierzykluses mit displayTimes() ausgeben kann

public static void kill():
Siehe Timer.kill

Package: exercises

Exercise.java

public Exercise(String className, String classCode, String testName, String testCode, boolean babystep, boolean timetracker, String exerciseName, String description):
Konstruktor: Konstruiert ein Exercise für die ArrayList mit dem int time

public Exercise(String className, String classCode, String testName, String testCode, boolean babystep, boolean timetracker, int time, String exerciseName, String description):
Konstruktor: Konstruiert ein Exercise für die ArrayList ohne dem int time