Systembeschreibung

1 Oberfläche

Die Main erstellt zunächst einen FileWriter für chart.txt, erstellt eine Scene mit sample.fxml und bindet tddt.css ein. Zudem startet sie das Programm und übergibt die primaryStage an TDDTMenu.
sample.fxml ist im Großen und Ganzen einerseits für das Layout und den Zustand (Größe, Position, disabled, editable etc.), andererseits für die aufzurufenden Funktionen der Buttons/MenuBar/... verantwortlich (z. B. beim btnextstep: onAction="#next").
tddt.css legt die Farben des Status und der zu highlightenden Strings in den TDDTextAreas (RichText) fest.

2 Hauptfunktionen

Bei TDDTMenu werden u. a. die von sample.fxml aufzurufenden Funktionen implementiert: open, chooseExercise, saveClick, close, next, previous, help (und showChart, die aber erst in Kapitel 3 aufgegriffen wird).
In den folgenden Unterkapiteln werden die Erläuterungen aufs Wesentliche reduziert. Wie enum funktioniert, wird einmalig in close (gui-package) erklärt.

2.1 catalog-package

Exercise: Es werden hauptsächlich Instanzvariablen für den Namen der Exercise, ihre Aufgabenbeschreibung und Inhalt und Name der Programm- sowie der Test-Datei der Exercise erstellt, denen man mit den add-Funktionen etwas zuweist, sodass man mit den get-Befehlen arbeiten kann.

Catalog: Catalog verwaltet eine ArrayList<Exercise> und besteht größtenteils aus load-Befehlen, um die Daten des Catalog mit den Exercises in die GUI zu laden.

CatalogParser: parse und convertToFileURL gibt mithilfe von SAXParser und XMLReader einen Catalog zurück.
XXXXXstartElement und der Rest fehlt nochXXXXX

2.2 gui-package

open:

- wird ausgeführt durch Klick auf miopen (Datei → Öffne Katalog)
- durch FileChooser öffnet sich ein Fenster, aus dem man eine XML-Datei auswählen muss
- Catalog catalog mit Exercises wird durch CatalogParsers parse in TDDListView<Exercise> (diese ist wie eine ListView) hineingeladen durch loadInListView
- File directory bekommt das ausgewählte File zugewiesen

chooseExercise:

- wird ausgeführt durch Klick auf eines der Elemente von der TDDListView

- der Text der Test- und Programm-Datei von der File directory werden in die jeweiligen TDDTextAreas (ähnlich einer TextArea) geladen mit einem loadExercise von catalog

saveClick:
- wird ausgeführt durch Klick auf misave (DateiSpeichern)
- ruft save auf
- save schreibt den zu speichernden Text in eine JAVA-Datei

close:
- wird ausgeführt durch Klick auf miclose (DateiProgramm schließen)
- ruft speichernAbfrage mit dem enum TriggerSaveOption.Close auf:
  speichernAbfrage ruft nun ein Alert-Fenster mit zwei Buttons (Ja/Nein) auf, das, je nachdem, welches TriggerSaveOption-enum übergeben wurde, einen anderen Text anzeigt
- primaryStage.close schließt das Fenster

next:
- wird ausgeführt durch Klick auf btnextstep (Button „Nächster Schritt“)
- ruft CompilerInteraction.compile auf, übergibt dabei einige Parameter

previous:
- wird ausgeführt durch Klick auf btback (Button „Schritt zurück“)
- ruft CompilerReport.back auf, übergibt dabei einige Parameter

help:
- wird ausgeführt durch mihelp (HilfeBenutzerhandbuch)
- öffnet help.html (Benutzerhandbuch)

2.3 compiler-package

CompilerInteraction:
Nachdem CompilerInteractions compile durch TDDTMenus next aufgerufen wird, wird das Programm oder der Test kompiliert mithilfe von CompilationUnit, JavaStringCompiler und Collection<compileError>.
Gibt es keine Kompilierfehler, werden die Testergebnisse (Anzahl der erfolgreichen Tests usw.) angezeigt durch CompilerReports showTestsResults. Außerdem wird durch continueable geprüft, ob CompilerReport mit changeReport die Phase wechseln soll.
Wenn es Kompilierfehler gibt, werden sie durch CompilerReports showErrors angezeigt.

CompilerReport:
changeReport erkennt, in welcher Phase (RED, GREEN, REFACTOR CODE, REFACTOR TEST) das Programm momentan ist, und wechselt die Phase bei Aufruf. Weiterhin bestimmt es, ob btback anklickbar ist, und legt durch TDDController fest, welche TDDTextArea editierbar ist und welche nicht.
back wird durch TDDTMenus previous aufgerufen während der Phase GREEN. Die Funktion versetzt das Programm zu dem Zustand, zu dem es am Anfang der Phase RED war.
showErrors arbeitet mit den beiden enums CompilerTarget und ErrorType, um zu bestimmen, welche Fehlermeldungen angezeigt werden sollen. Dabei arbeitet es mit CompileTarget.Test (target) und einem ErrorType, den es mit der error-Funktion, ebenfalls in CompilerReport zu finden, erstellt.
showTestResults setzt ein paar Strings zusammen und arbeitet mit TestResult.

3 Zusatzfunktionen

Im TDDTMenu werden ein Babysteps baby und ein Tracking tracker deklariert, die bei chooseExercise instanziiert werden, falls die isBabysteps/isTracking-Funktion von der momentanen Exercise currentExercise true zurückgibt. Ob isBabysteps/isTracking true zurückgibt und welche Zeit bei Babysteps eingestellt ist, kann man im Übungskatalog einstellen (z. B. catalogsample.xml).

3.1 Babysteps

Ab der Instanziierung beginnt der Timer t zu laufen. Babysteps' Konstruktor bekommt currentExercise und die TDDTextArea tatest übergeben, um nach Ablauf des Timers t die Zeit (getBabystepTime von currentExercise gespeichert in int timer) und beim ersten Ablauf die TDDTextArea zurücksetzen zu können.
babystep bekommt die TDDLabel lbstatus und lbtime sowie die TDDTextAreas taeditor und tatest übergeben, um die Zeitanzeige zu ändern und, je nach Status, taeditor bzw. tatest zurückzusetzen.

Klickt man btnextstep, wird bei CompilerReports changeReport, falls es zuvor durch CompilerInteractions compile mit baby als einer der Übergabeparameter aufgerufen wird, refreshTimer, getOldEditor beim Wechsel von RED zu GREEN oder beim Wechsel von REFACTOR TEST zu RED getOldTest aufgerufen, damit man beim nächsten Ablauf des Timers taeditor und/oder tatest zurücksetzen kann, da man den Text durch die beiden get-Funktionen speichert.
Klickt man btback vor Ablauf der Zeit, verhält sich das Programm genau wie bei ausgeschaltetem Babysteps, es kommt lediglich hinzu, dass refreshTimer aufgerufen wird, sodass die Zeit zurückgesetzt wird.
Wechselt man die Exercise, wird der alte Timer gestoppt in chooseExercise.

3.2 Tracking

Für Tracking wurde eine eigene Klasse Timer erstellt. Dieser beinhaltet einige Methoden, um ein Zeitintervall zu bestimmen, indem man einen Timer startet, stoppt und dann die Differenz bildet.
Tracking selbst besitzt eine ArrayList<Timer> mit vier Einträgen für alle Status (RED, GREEN, REFACTOR CODE, REFACTOR TEST).

Klickt man btnextstep, wird bei CompilerReports changeReport, falls es zuvor durch CompilerInteractions compile mit tracker als einer der Übergabeparameter aufgerufen wird, je nach Status einer der Timer gestoppt und der nächste gestartet.
Des Weiteren wird ein zuvor erstelltes int-Array chart genutzt, um die Zeiten zu speichern, damit sie im Kuchendiagramm angezeigt werden können. Es ist zu finden unter Statistiken → Benutzeranalyse. Die Implementierung des Diagramms ist in TDDTMenu zu finden unter showChart, dazu wurde die Klasse PieChart benutzt.

3.3 RichText

XXXXXfehlt nochXXXXX

3.4 ATDD

XXXXXfehlt nochXXXXX