TDDT-Bericht

Klasse Analyse:
Die Klasse stellt die Grundlage für die Phasenanalyse dar, welche man in der MenüBar unter Tracking finden kann. Sie speichert die Zeit, die der Nutzer in den verschiedenen Phasen verbracht hat und erstellt daraus ein Tortendiagramm.

Klasse Aufgabe:
Mit der Klasse Aufgabe kann man ein Objekt Aufgabe erstellen. Eine Aufgabe beinhaltet immer einen Namen (als String), eine Beschreibung (auch als String), ein Inhaltsarray für Klassen und eines für Interfaces, außerdem ein Objekt Inhalt für Test und einen Boolean Babystep, welcher angibt ob die Aufgabe in Babystep gelöst werden soll. Auch in dieser Klasse kann man auf alle Inhalte einfach mit den entsprechenden "getter" Methoden zugreifen. Diese sind in diesem Fall: getName, getBeschreibung, getKlassen, getInterfaace, getTest, getValueBabystep.

Klasse Inhalt:
Mit der Klasse Inhalt, wird ein Objekt Inhalt erzeugt. Dieses hat einen Namen und einen Text.
Auf den Text und den Namen kann man einfach mit den "getter" Methoden getName und getText zugreifen.

Klasse CodeTester:
Die Klasse CodeTester ist dafür verantwortlich, den in der Phase „green“ entwickelten Code zu testen, Fehler zu ermitteln, Ausgaben anzuzeigen und den Fortschritt in externen Dateien zu speichern.
Außerdem werden hier alle Änderungen (in der „green“-Phase) dem Tracker mitgeteilt.

Um die Ausgaben des Programms zu ermitteln, wird erst der gesamte Code in eine externe Datei geschrieben (normalerweise werden in der Klasse TestTester schon die richtigen Unterverzeichnisse angelegt, da dies die Phase ist, in der das Programm startet bzw. in der man beginnt, wenn man eine neue Aufgabe bearbeiten möchte, dennoch wird beim schreiben der externen Dateien nochmals geprüft, ob das richtige Verzeichnis angelegt ist) und im Falle einer Syntaxfehler-freien Implementierung mittels einer batch-Datei ausgeführt. Alle Ausgaben der batch-Dateien werden abgefangen und auf die virtuelle Konsole weitergeleitet.
Sind Fehler im Programm, die der Compiler aus der benutzten Bibliothek erkennt, werden diese mit entsprechender Beschreibung auf die virtuelle Konsole geschrieben.

Für die Wechsellogik gibt es das Attribut getestetUndFehlerfrei, welches genau dann auf true gesetzt wird, wenn der Code kompiliert wurde und es keinen (Syntax-)Fehler gab.

Der CodeTester bekommt den geschriebenen Code als einen großen String übergeben; im Falle eines objektorientierten Programms werden also nicht alle Klassen einzeln getestet, sondern die Hauptklasse, in der die Main-Funktion zu finden ist, als public class deklariert und alle weiteren Klassen als class deklariert. Somit kann der Compiler aus der virtual-kata-lib auch Beziehungen zwischen zwei (oder mehr) Klassen verstehen.

Hauptklasse Main

Die Hauptklasse ist für den Programmstart und -ablauf zuständig. Wenn der Nutzer das Programm zum ersten Mal startet (in der config.txt also kein Eintrag für den gewählten Katalog vorhanden ist) wird das Programm mit einem Begrüßungsfenster gestartet (initialStart()). Hat der Nutzer bereits einen Katalog ausgewählt, oder tut dies über das Begrüßungsfenster so wird das Hauptprogramm gestartet.
Dazu werden die entsprechenden .fxml Dateien und die dazugehörigen Controller geladen und erstellt. Ebenfalls wird hier die Liste an Aufgaben, die man oben in der MenüBar finden kann, mit Inhalten aus dem gewählten Katalog befüllt. Ist das Hauptprogramm gestartet, so sind erst einmal alle Elemente auf der Oberfläche unscharf geschaltet, bis der Nutzer eine Aufgabe zum Bearbeiten aussucht.
Wird die Aufgabe gewechselt oder das Programm beendet wird über diese Klasse gefragt, ob die Zwischenergebnisse gespeichert werden sollen. Falls ja, werden in den Klassen CodeTester, Tracker und TestTester die Methoden ausgelöst, welche den aktuellen Zwischenstand der Aufgabe in externen Dateien speichert.
Beim Wechsel der Aufgabe wird von hier aus außerdem der Name der neuen Aufgabe weitergeleitet, sodass die anderen Klassen wissen, in welchem Verzeichnis sie ab jetzt arbeiten.
Die Funktion getCode() ist dafür zuständig, falls eine Aufgabe schon einmal bearbeitet wurde (ein Verzeichnis mit dem Aufgabennamen also existiert), den Code aus der gespeicherten Datei zu lesen und eine HashMap zu erstellen, die für jede Klasse (Key) den Code beinhaltet (Value).
Über die Funktion aktualisiereAufgabe() wird der Oberfläche angewiesen, die neuen Inhalt der Aufgabe zu laden.
Beim Beenden der Aufgabe wird außerdem die config.txt mit dem Pfad zum Aufgabenkatalog und dem Namen der zuletzt bearbeiteten Aufgabe befüllt.

Klasse TestTester:
Wie der Name schon sagt, testet diese Klasse die Tests des Nutzers auf Bestehen und Programmierfehler.
Zudem speichert Sie die Tests ab, damit man später weiterarbeiten kann.

Methoden von TestTester:
testeTests: testet die Tests
writeTest: erzeugt die Datei und speichert den Test
logging: gibt Informationen an den Tracker weiter
fehlerString: Funktion zum ausgeben der Programmierfehler
Rueckgabe: Funktion zum ausgeben wie viele Tests durchlaufen und wie viele "falsch" sind
setKonsolenTest: gibt Text in die Konsole aus
setTracker: setzt den Tracker
setLetzterStandCode: wird fürs logging benötigt
getCorrectPath: setzt den Dateipfad
setNameAufgabe: setzt den Namen des Tests anhand des Aufgabenamens

Klasse Tracker:
Die Klasse Tracker ist für das Erstellen der Phasenanalyse und das loggen der Ereignisse zuständig. Es werden zum einen alle Phasenwechsel geloggt, damit man sehen kann, wie viel Zeit man für das Bearbeiten der Phase im letzten Durchgang benötigt hat und zum anderen werden jegliche Änderungen, die der Nutzer an seinem Programm (Code oder Tests) vornimmt mit einer Zeitangabe und falls fehlerhaft mit dem Fehlercode geloggt.
So kann jederzeit über die Phasenanalyse ermittelt werden, welche Fehler aufgetreten sind, in welcher Phase diese aufgetreten sind und wann Änderungen vorgenommen wurden.
Die Funktion ermittleNeuerung ist dafür zuständig, Unterschiede zwischen dem neuen Code und dem alten Code festzustellen.
Dabei untersucht sie die beiden Texte zeilenweise und kontextunabhängig, sodass nicht immer zusammenhängende Blöcke bei den Änderungen erkennbar sind.
Wird das Programm beendet oder die Aufgabe gewechselt und der Nutzer möchte seine Zwischenstände speichern, so wird eine trackerstand.txt erstellt, welche Informationen über den aktuellen Trackerstand beinhaltet, sodass später, wenn die Aufgabe weitergeführt wird, der Stand entsprechend sinnvoll weitergeführt werden kann.

Klasse XMLParser:
Der XMLParser liest den im Programm ausgewählten Katalog ein und gibt am Ende ein Aufgabenarray zurück. Das Aufgabenarray bekommt man durch die Methode getAufgaben.

Diese geht alle im Katalog gegeben Aufgaben durch und sucht sich von jeder einzelnen im Katalog festgelegten Aufgabe den Namen, die Beschreibung, alle Klassen (daher braucht man an dieser Stelle im Objekt Aufgabe auch ein Array), alle Interfaces (daher braucht man an dieser Stelle im Objekt Aufgabe auch ein Array), den Test und den Wert von Babystep heraus. Somit hat man am Ende der Methode alle Aufgaben aus dem entsprechenden Katalog in einem Array und kann dank Inhalt und Aufgabe jederzeit auf alles zugreifen, wodurch sich die jeweiligen Felder in der Benutzeroberfläche leicht ausfüllen lassen.

Controllerklassen:

Klasse OberflaechenController:
Der OberflaechenController ist die Controllerklasse zu Oberflaeche.fxml und enthält deshalb verschiedene Methoden, die zur Interaktion der anderen Klassen mit der Benutzeroberfläche nötig sind.

Die Klasse OberflaechenController verwaltet die meisten der Oberflächeninhalte (außer Menü und Erster-Start-Fenster) und unter anderem die durch die zugehörige .fmxl festgelegten Methoden zur Interaktion mit den dort beschriebenen Teilen der Benutzeroberfläche. Sie stellt zusätzliche, nicht in der .fxml-Datei festgehaltene Funktionen, wie z.B. das An- und Ausschalten der Test- und Code-Sektion, zur Verfügung. Der OberflaechenController interagiert mit den Klassen: Aufgabe, CodeTester, TestTester, Tracker und Main.

Klasse StartbildschirmController:
Der StartbildschirmController ist die Controllerklasse zu Startbildschirm.fxml und enthält die Methoden für die drei Schaltflächen des Startbildschirms: “Katalog auswählen”, “Handbuch anzeigen” und “Programm beenden”.

Klasse HauptfensterController:
Die Klasse HauptfensterController ist die Controllerklasse zu Hauptfenster.fxml und stellt die Methoden für die verschiedenen Punkte der Menüleiste des Hauptfensters bereit. So gibt es z.B eine Methode handleMenueEinstellungen zum Öffnen des Einstellungsfensters oder eine Methode handleGebrauchsanweisung, welche das Handbuch anzeigen lässt usw.