Bericht für das Projekt 7: TDDT

Allgemeiner Programmaufbau:

TDDT:

- **Editor**
- **Compiler**
- **XML_Body**
- **Erweiterungen**

Editor

Klasse main:

Der Klasse main lädt die fxml sample.fxml und zeigt diese in einer szene an.

Methoden:

Start(Stage primaryStage):

- Lädt die fxml sample.fxml nach FXMLLoader
- Setzt Controller (hier vorgegeben durch sample.fxml) für FXMLLoader
- Setzt und zeigt die Stage

sample.fxml

Bestimmt Layout der UI und legt den Controller (hier Controller.java) fest

Klasse IntSenderModel

Der zweck dieser Klasse ist das Austausch vom Nummer der bei getExersice() gewählten Projekts mit Model.java

Methoden:

- public Integer getNumber()

  gint die Nummer

- public void setNumber(int number)

  setzt die Nummer

- public IntegerProperty getProperty()

  liefert die Property von Nummer

Klasse Model

Ist ein Wrapper für Exersises und liefert diese untsprechend der Nachfrage an Controller

Methoden:

- public Model(int index)

  Initialisiert Model und Lädt sofort den Projekt

  mit Nummer index

- public Exersise getExersise()

  Liefert den Projekt

- private Exersise loadExersise()

  Lädt den Projekt mit index index

- public void saveExersise()

  Speichert den Projekt an der Stelle index

- public ArrayList<TextFile> getAllTextFiles(TextFile code, TextFile test, TextFile task)

  Liefert Code, Tests und Aufgabe als Liste der

  gewrappter(mit TextFile) Texte

- public Boolean isBabystepsEnabled()

  Liest aus den xml, ob Babysteps aktiviert sind

- public int getBabyStepTimer()

  liefert den Wert für Babysteps

Klasse TextFile

TextFile ist ein Wrapper für die Listen mit Inhalt der Aufgabe/Code/e.t.c

Methoden:

- public TextFile(List<String> content)

  Initialisierung von TextFile

- public List<String> getContent()

  Liefert den Inhalt zurück

  (also eine Liste mit Strings)

- public ArrayList<String> getAsArrayList()

  Ist eine weitere Methode zum

  setten vom Inhalt.

  Ist rein für Lambdas da

Klasse Controller

Ist der Controller für sample.fxml

Methoden:

- public Controller()

  Initialisierung vom Controller. Hier werden

  Modelle sowie Compier initialisiert

- private void makeStep()

  macht eine Aktion entsprechend der Phase vom

  Projekt. Z.B. hat man gerade Tests geschrieben,

  so werden mit der makeStep der Code und die Tests

  commpiliert und es wird geschaut, ob die

  Bedingungen erfüllt sind. D.h. sollten zwei Tests

  nicht bestanden sein, heißt das, dass der Nutzer

  mehr als ein Test auf einmal geschrieben hat, was

  gegen die Regeln ist. Sollte doch nur ein Test

  durchfallen, so ist alles in Ordnung und der

  Nutzer darf nun den Code bearbeiten.

  Es gibt drei Phasen: 0 == Test wird geschrieben

  1 == Code wird geschrieben

  2 == code wird refactored

- public void stepBack()

  sollte Nutzer ein Fehler z.B. beim aufstellen von Tests

  gemacht haben, hat er die Möglichkeit, zurück zu gehen, und

  dies zu beheben

- private void onLoad()

  Hiermit wird der Projekt mithilfe von getExercise() gewählt

  und dann geladen

- private void setColorAccordingToPhase()

  setzt die Farbe von editierbarem Fenster entsprechend der Phase

- private void getExersice()

  mach ein neues Fenster auf und lässt den Nutzer den Projekt

  aus der Liste wählen

- private void setAllTextFiles(ArrayList<TextFile> t)

  gibt neuen Wert an code/test/task

- private void onSave()

  speichert das Projekt
- private void onClose()

  Schließt das Programm
- private void onAbout()

  about-Fenster
- private void StartBabysteps()

  startet Babysteps, falls möglich ist
- private onATDDT()

  öffnet ein neues Fenster, um die ATDDT Tests zu schreiben

Compiler

Klasse Task

Der Klasse Babysteps wird die Anzahl der Sekunden für den Timer, der Controler des Fensters zum zurücksetzen des Codes und das Label zum Anzeigen des Countdowns, übergeben.

Die Klasse erbt von TimerTask, damit diese für einen Timer genutzt werden kann.

Die Klasse ist nur für die Klasse Babysteps sichtbar.

Methoden:

- void run():
  Zählt die Sekunden um eins runter und schreibt die Sekunden für den Nutzer sichtbar in das Label auf dem Fenster oben rechts. Sind die Sekunden runter auf 0 gezählt, wird die Methode void stepBack() des Controlers ausgeführt, damit der Nutzer nicht mehr im Code schrieben kann, sondern die Tests ändern kann.

Klasse CompilerHelper

Diese Klasse soll helfen mit der Bibliothek Virtual-Kata von Dr. Jens Bendisposto zu arbeiten. Dafür stellt diese Klasse mehrere Methoden zur Verfügung, die eine Überprüfung von Kompilierfehlern für die Tests oder den Quellcode und zur Feststellung, von welcher Testklasse Tests nicht bestanden wurden.

Methoden:

- void SetFeatureTest(String feautureTestClassName, String featureTestClassSource):
  setzen der Klasse des Akzepttanztest

- void SetTest(String testClassName, String testClassSource):
  setzen der Klasse für normale Tests im Ablauf

- void AddSourceClass(String className, String classSource):
  setzten der Klasse, der den Code zum löse der Tests enthält

- void CompileAndTest():
  Kompiliert die Klassen (auch die Testklassen) die dem CompilerHelper bereits übergeben wurden

- TestResult GetTestResult():
  übergibt das Objekt der Ergebnisse für die Tests
  TestResult ist eine Klasse der Bibliothek „virtual kata“

- CompilerResult GetCompilerResult():
  übergibt das Objekt der Ergebnisse für das Kompilieren aller Klassen
  CompilerResult ist eine Klasse der Bibliothek „virtual kata“

- Boolean HasCompilerErrors():
  prüft nach ob Fehler beim Kompilieren aufgetreten sind und übergibt dann das Ergebnis

- String GetSourceClassCompilerError():
  übergibt den Fehler der beim Kompilieren des Codes zum Erfüllen der Tests aufgetreten sind
  Nutzt GetCompilerError(…)

- String GetTestClassCompilerError():
  Übergibt den Fehler der beim Kompilieren, der Testklasse aufgetreten sind
  Nutzt GetCompilerError(…)

- String GetFeatureTestClassClassCompilerError():
  Übergibt den Fehler der beim Kompilieren, der Akzeptanztestklassse aufgetreten sind
  Nutzt GetCompilerError(…)

- Private String GetCompilerError(CompilationUnit cu):
  CompilationUnit ist eine Klasse der Bibliothek „virtual kata“
  Sammelt alle Kompilierfehler in einen String für die genannte Klasse in dem Objekt cu

- String GetCompilerErrors():
  gibt alle Fehler zurück, die beim Kompilieren aufgetreten sind, von allen Klassen

- String GetTestFaillures():
  sammelt von allen Tests, die fehlgeschlagen sind, die Message von junit und gibt sie zurück

- int NumberOfSucceddfulTests():
  gibt die Anzahl der nicht fehlgeschlagenen Tests zurück

- int NumberOfFailedTests():
  gibt die Anzahl der nicht bestandenen Tests aus der Testklasse zurück
  Nutzt CountFailedFeatureTest(…)

- int NumberOfFailledFeatureTest():
  gibt die Anzahl der nicht bestandenen Tests aus der Testklasse zurück
  Nutzt CountFailedFeatureTest(…)

- private int CountFeiledFeatureTest(CompilationUnit cu):
  CompilationUnit ist eine Klasse der Bibliothek „virtual kata“
  zählt alle fehlgeschlagenen Tests für die genannte TestKlasse in dem Objekt cu

XML_Body

Klasse Exercises

Schnittstelle/Gerüst für xml Obertyp von Exercise. Wird im Model als ganzes geladen.
Eigentlich nur eine Liste aus „Exercise“ Klassen

Arbeitet mit:

- Exercise
- Model
- XMLController

Klasse Exercise

Schnittstelle/Gerüst für XML Untertyp von Exercises. Wird im Model als Teil von Exercises geladen. Besitzt diverse Funktionen um die vom XMLController kommenden Strings in für das Oberflächliche Programm benutzbare Werte umzuwandeln, und den Zugriff auf jene ermöglichen.

Arbeitet mit:

- Exercises
- Model
- XMLController

Klasse XMLController

Ist für das tatsächliche Speichern und Laden der XML Dateien verantwortlich. Greift dabei folglich auf ein XML-Dokument zu. Liest das XML-DOKUMENT und konstruiert mit diesen eine neue Exercises Klasse.

Arbeitet mit:

- Exercises
- Aufgaben.xml
- Exercise

Klasse Model

Ein Versuch die Logik aus dem FXML-Controller ein wenig auszulagern. Hier werden die hauptsächlich relevanten Ladevorgänge ausgeführt. Hier kommen außerdem die XML-Klassen zusammen.

Arbeitet mit:

- Exercises
- Exercise
- XMLController

Klasse Aufgaben.xml

XML-DOKUMENT für den Controller. Hier werden Änderungen abgespeichert und ggf. geladen.

Arbeitet mit: XMLController

Erweiterungen

Klasse Babysteps:

Der Klasse Babysteps wird die Anzahl der Sekunden für den Timer, der Controler des Fensters zum zurücksetzen des Codes und das Label zum Anzeigen des Countdowns, übergeben.

Methoden:

- void Start():
  Starten des Timer. Der Timer ruft einmal pro Sekunde die geerbte Methode void Run von dem Task-Objekt auf.

- void Stop():
  Stopen des Timers

- boolean Running():
  gibt an, ob der Timer noch läuft

Konstruktor:

Der Konstruktor erstellt das Task-Objekt (erbt von TimerTask), dass dem Timer übergeben wird bei void Start(). Dem Task-Objekt, werden die Sekunden, der Controler und das Label übergeben.