## 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 (also eine Liste mit Strings) zurück

* + - 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