Mastermain:

main():
Lädt und startet die GUI.fxml Datei und startet somit das Programm.

Controller:

initialize():
-startet die menuGUI,
-setzt je nach Einstellungen die entsprechende Startphase,
-lässt den Presetdeliverer.main() einmal durchlaufen,
-fügt den Java- und Testcode in die entsprechenden Textfelder ein,
-startet je nach Einstellungen den Babysteptimer,
-speicher den Java- und Testcode einmal ab, um die finalen Dateien zu erstellen.
- ruft "managephasegui" auf, um die GUI entsprechend der Phase anzuzeigen.

managephasegui():
Aktiviert/deaktiviert alle grafischen elemente der GUI je nach Phase.
Stopt/startet den Babysteptimer je nach Phase.

check():
Check wird durch den "check" Button aufgerufen. Zuerst werden die Textfelder ausgelesen und aus der PresetDataBase der Klassenname des Codes, sowie der Klassenname des Testes ausgelesen. Als nächstes werden zwei Compilation Units erstellt (Test und Code). Im nächsten Schritt werden beide Compilation Units mithilfe eines Compilers kompiliert. Die Ergebnisse der Kompilierung werden gespeichert. Check() prüft nun ob Kompilierungsfehler vorhanden sind, wenn ja werden die Fehlermeldungen mit Zeilenangabe und genauer Beschreibung des Fehlers im Error Feld der GUI ausgegeben. Dies gilt sowohl für Test als auch für den Code.
Sollten keine Kompilierungsfehler aufgetreten sein, prüft Check ob Tests fehlgeschlagen sind. Check() gibt die Fehleranzahl aus, sowie die genauen Testmethoden, tatsächlichen und ausgegebenen Werte. Das weitere Vorgehen von Check() ist abhängig von der Phase.

1. In Phase eins (Test schreiben), muss genau ein Test fehlschlagen. Sollte das der Fall sein wird die Save Testmethode aufgerufen und die Phase auf 2 gesetzt. Zudem wird babyclock.reset() aufgerufen, sowie managephasegui(phase) mit der nun geänderten Phase. Abschließend wird das Error Feld in der GUI geleert.
2. In Phase zwei (Code schreiben) darf kein Test fehlschlagen. Wenn diese Bedingung erfüllt ist wird savecode() aufgerufen. Die Phase wird auf drei gesetzt und babyclock.reset() wird wieder aufgerufen. Danach wird managephasegui(phase) mit der nun aktuellen Phase drei aufgerufen. Anschließend wird das Error Feld in der GUI bereinigt.
3. In Phase drei (Refactoring) darf wieder kein Test fehlschlagen. Check prüft nun ob ATDD aktiviert ist. Wenn ja werden die benötigten Daten für die ATDD Methoden entsprechend gespeichert. Das entsprechende ATDD Fenster wird aufgerufen. Check pausiert nun bis das ATDD Fenster geschlossen wurde. Anschließend resetiert Check die babyclock, ändert die Phase auf eins ab und ruft managephasegui(phase) auf. Wenn ATDD nicht gesetzt ist ruft Check savecode() auf, setzt die Phase auf eins und resetiert die babyclock. Abschließend wird managephasegui(phase) aufgerufen, sowie das Error Feld der GUI geleert.

goBack(): Wird beim betätigen des „back“-Buttons aufgerufen. Falls die Phase gerade 2(schreibe Code) ist kommt man zurück in die Testphase. Falls die Phase gerade 3(Refactor) ist wird der Code zurückgesetzt.

backandcheck(): Setzt Testfeld, Codefeld und Timer zurück.

Menucontroller:

createpreset():
Erstellt die "createdpreset"-Datei mit den in der menuGUI eingegebenen Daten und setzt den PresetDataBase.presetpath auf die entsprechende Datei.

loadcodefile():
Eröffnet einen Filechooser und und setzt PresetDataBase.codefilepath auf den Pfad der gewählten Datei.
Aktiviert den "continue"-Button, falls ein code- und presetfile Pfad angegeben wurde.

loadpresetfile():
Eröffnet einen Filechooser und und setzt PresetDataBase.presetpath auf den Pfad der gewählten Datei.
Aktiviert den "continue"-Button, falls ein code- und presetfile Pfad angegeben wurde.

goon():
Setzt die Namen der geladenen/erstellen Dateien in die PresetDataBase ein und schließt anschließend das Fenster.

ATDDController:

Das ATDD-Feature kann je nach Präferenz aktiviert oder deaktiviert werden. Wenn es aktiviert ist wird beim Start ein neues Fenster geöffnet in dem der ATDD-Test geschrieben werden soll. Falls es deaktiviert ist wird mit dem ganz normalen TDD Zyklus begonnen. Bei aktiviertem ATDD-Feature wird nach jedem Refactoringschritt geprüft ob die ATDD-Tests fehlerfrei durchlaufen. Falls dies der Fall ist wird das ATDD-Fenster wieder geöffnet. Die geschieht mit Hilfe der ATDDFailedTests() Methode. Der gesamte Code wird kompiliert, falls dabei Fehler auftreten werden diese im Errorfeld angezeigt. Wenn keine Compile-Errors aufgetreten sind werden die Tests ausgeführt. Bei keinem fehlschlagenden Test wird das ATDD-Fenster geöffnet. Ansonsten wird die Anzahl der fehlschlagenden Tests ausgegeben. Nachdem man weitere ATDD-Tests geschrieben hat betätigt man den check-Button welcher wieder ATDDFailedTests() aufruft und nur in die nächste Phase springt, wenn genau ein Test fehlschlägt.

JavaToEditor:
JavaToEditor(String): Erzeugt ein Objekt welches einen Pfad übergeben bekommt.
Read(): Liest den Inhalt der am Pfad liegenden Datei aus und gibt sie als String zurück

EditorToJava:
EditorToJava(String): Erzeugt ein Objekt welches einen Pfad übergeben bekommt.
Save(String): Bekommt einen String übergeben und erzeugt bzw. schreibt in die im Objekt hinterlegte Datei. Diese wird als .java File gespeichert.

TxtToJava:
transform():
Lädt eine .txt Datei ein und speichert diese als .java Datei.

XMLReader:

Der XMLReader stellt die Schnittstelle zwischen Presetsetter und PresetDataBase dar.Außerdem ließt er die die Presetdatei.txt ein und gibt die Zeilen welche einen Wert für die Lücke der Codevorlage besitzen weiter an den Presetsetter. Der Inhalt der Zeile wird vom XMLReader in die PresetDataBase geschrieben.

Babysteps:

Die Klasse BabystepClock ist zu realisierung der Babystepfunktion zuständig. Von der Klasse werden BabystepClock Objekt instanziiert, beim Erstellen eines Objekts wird auf die PresetDataBase zugegriffen und die Maximalzeit des Babysteptimers abgefragt, und ein neuer Thread erstellt. In dem Thread wird dann jede Sekunde ein Zähler erhöht, solang bis der Maxwert der in den Presettings angegeben wurde erreicht wird.

Die Klasse liefert ein paar Methoden um mit der BabystepsClock zu interagieren.

- stop() um das hochzählen des Zählers zu stoppen.

- reset() um den Zähler zurück zu setzen.

- restart() setz den Zähler wieder zurück und startet das Hochzählen den Zählers.

Somit existiert über den gesamten Programmverlauf nur ein BabystepClock Objekt.

Presetdeliverer:

Beim Programmstart wird der Presetdeliverer aufgerufen. Er enthält die Codevorlagen für die ersten Java Dateien. Nachdem durch den XMLReader die Werte für das Füllen der Lücken gefunden wurden fügt der Presetdeliverer die fehlenden Werte in die Codevorlage ein und schreibt die Javatest Datei. Nach befüllen der Lücken in der Javacode Datei ruft der Presetdeliverer die Methode TxtToJava.transform() auf welche dann die Javacode Datei schreibt.

Presetsetter:

Der Presetsetter bekommt vom XMLReader die Zeilen welche Namen und Werte enthalten übergeben, und sucht die Stelle heraus welche den konkreten Wert bzw. Namen enthält und gibt diesen zurück. Ausnahme ist wenn bei der Babystepclock, der Zeitwert wird berechnet und sofort in die PresetDataBase geschrieben.

PresetDataBase:

Die PresetDataBase ist die Schnittstelle der eingelesen Werte zu den anderen Klassen und der GUI. Sie wird vom XMLReader mit den Werten beschrieben.